Graphical Interactive Control Design and Implementation Environment

Contract No. NAS1-20201

Final Report for the Period
1 May 1994 – 30 April 1996

Prepared for:

National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681 - 0001

Prepared by:

Dr. Robert L. Kosut
Dr. Ywh-Pyng Harn
Integrated Systems Inc.

ISI Report No. 7410-24

June 25, 1996

Graphical Interactive Control Design and Implementation Environment

Contract No. NAS1-20201

Final Report for the Period
1 May 1994 – 30 April 1996

Prepared for:

National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

Prepared by:

Dr. Robert L. Kosut
Dr. Ywh-Pyng Harn
Integrated Systems Inc.

Approved by:

Robert L. Kosut

Robert L. Kosut, Director
Control Systems Research Dept.

ISI Report No. 7410-24

June 25, 1996

Summary

Current commercial CACE (Computer-Aided-Control-Engineering) environments and tools place some severe restrictions on the user about the multi-objective control system design, which are major obstacles to the wider application of advanced control systems. In particular:

- current tools require the user to specify *indirect* controller constraints, e.g., weighting matrices or transfer functions ("design knobs"), and then check that the closed-loop system satisfies the actual requirements, e.g., overshoot. This approach requires many design iterations and considerable insight by the user.
- current tools require the user to have an intimate knowledge of the control design algorithms and the real-time controller hardware. As a result, a very high level of expertise is often required to make effective use of the tools.

To eliminate these obstacles, a CACE environment has been implemented in this project to meet the following objectives:

- The user is able to design controllers *directly* from closed-loop specifications.
- The user interacts with the environment through a GUI (Graphical User Interface) that eliminates the need for intimate knowledge of optimization and control design algorithms.
- The GUI capability exists to implement, interact, and modify the simulation model on an automation and on-line basis during the numerical or real-time simulation process.

In this new environment, the user's interaction stays at the level of design performance specifications. Design problems can be posed and solved – and if necessary, the controller can be tested through numerical simulation or implemented in real-time – all without the user being overloaded by the details of the underlying toolbox language, design and optimization algorithms, or hardware. The design environment integrates the design phase and the simulation phase of a design cycle together, and provides a complete and extensive treatment for practical design problems. In summary, with the new CACE tool called CODA (Convex Optimization based Design Algorithm), *a larger class of design problems can be solved in a more complete and extensive way by a broader range of users!*

Contents

1 Introduction **1**

1.1 Motivation and Objective 1

1.2 CODA: A New CACE Design Environment 2

1.3 Outline of Report 4

1.4 Current ISI CACE Environment 6

2 Design Approaches and Preprocessing Scheme **7**

2.1 4-Block Plant Design Model 7

2.2 Q (Youla) parametrization 8

2.3 Finite-Dimensional Expansion for Q parameter 11

2.4 Design Specifications 12

2.5 Problem Formulation 13

2.5.1 $\mathcal{H}_2$ Design Function 13

2.5.2 Step/Impulse Design Functions 14

2.5.3 MSV Design Function 15

2.5.4 Integral-Action Requirements 16

2.5.5 Conclusion 16

2.6 Preprocessing Scheme 17

2.6.1 Preprocessing for $\mathcal{H}_2$ Function 18

2.6.2 Preprocessing for Step/Impulse Responses 19

2.6.3 Special Case with Zero K and L 23

2.6.4 Conclusion 23

3 Optimization Algorithms **25**

3.1 Introduction 25

3.2 Descent-Direction Method 27

3.3 Barrier-Function Method 29

3.4 A Hybrid Method 33

4 Interaction with ISIM and RealSim **35**

4.1 Introduction 35

4.2 Automated Modeling of Controller and Feedback Systems via SBA . 37

4.3 Update of Controller Parameter via RVE 38

4.4 ISIM Results Appearing on GUI Windows via a UCI 39

4.5 Conclusion 40

5 Graphical User Interface **41**

5.1 Introduction 41

5.2 CODA Main Window 42

5.2.1 File Menu 45

5.2.2 Edit Menu 48

5.2.3 Design Menu 51

5.2.4 Simulation Menu 56

5.2.5 Special Menu 61

5.3 Interactive Function and Trade-off Windows 62

5.3.1 Interactive Function Windows 62

5.3.2 Multiple-Function Windows 66

5.3.3 Trade-Off Windows 69

References **73**

1 Introduction

1.1 Motivation and Objective

A typical controller synthesis problem is to design a controller for a given single plant in such a way that several requirements are satisfied. Obviously the most basic requirement is

- stability

i.e., the controller must stabilize the plant, or somewhat more generally, place all closed-loop poles in some prespecified region. Additional requirements include, but are not restricted to:

- robustness against unmodeled sector-bounded perturbations
- minimum and maximum bandwidth for the controlled system
- specifications of the rise time and settling time for the step response
- maximum instantaneous amplitudes for each of the plant inputs
- maximum RMS (root-mean-square) levels to noise inputs

and so on. Current commercial CACE (Computer-Aided-Control-Engineering) environments and tools, e.g., MATRIX$_X$, MATLAB and ICDM (Interactive Control Design Module) in MATRIX$_X$, place some severe restrictions on the user about the multi-objective control system design, which are major obstacles to the wider application of advanced control systems. In particular:

- current tools require the user to specify *indirect* controller constraints, e.g., weighting matrices or transfer functions ("design knobs"), and then check that the closed-loop system satisfies the actual requirements, e.g., overshoot. This approach requires many design iterations and considerable insight by the user.
- current tools require the user to have an intimate knowledge of the control design algorithms and the real-time controller hardware. As a result, a very high level of expertise is often required to make effective use of the tools.

To eliminate these obstacles, a CACE environment has been implemented in this project to meet the following objectives:

- The user is able to design controllers *directly* from closed-loop specifications.

- The user interacts with the environment through a GUI (Graphical User Interface) that eliminates the need for intimate knowledge of optimization and control design algorithms.
- The GUI capability exists to implement, interact, and modify the simulation model on an automation and on-line basis during the numerical or real-time simulation process.

In this new environment, the user's interaction stays at the level of design performance specifications. Design problems can be posed and solved - and if necessary, the controller can be tested through numerical simulation or implemented in real-time - all without the user being overloaded by the details of the underlying toolbox language, design and optimization algorithms, or hardware. The design environment integrates the design phase and the simulation phase of a design cycle together, and provides a complete and extensive treatment for practical design problems. In summary, with the new CACE tool, *a larger class of design problems can be solved in a more complete and extensive way by a broader range of users!*

1.2 CODA: A New CACE Design Environment

The new CACE design environment is refered to as **CODA**, an acronym for Convex Optimization based Design Approach. It can be utilized for both continuous-time and discrete-time linear MIMO (multi-input-multi-output) system design. CODA is based on two facts:

1. All stable closed-loop transfer functions can be expressed as an affine function of a stable transfer function called the Youla parameter or the Q parameter;
2. many common design specifications are convex functions of the closed-loop transfer functions, and hence are convex functions of the Q parameter.

The basic architecture of CODA is shown in Figure 1.1 where it can be observed that the user interacts with CODA only through the GUI in both the design phase and the simulation phase. The other functionalities such as preprocessing and optimization are completely transparent to the user. The interaction of the design phase with the simulation phase is clearly described in Figure 1.1.

The flow chart of a typical design cycle is shown in Figure 1.2 where the dashed boxes indicate that the functionalities therein are totally transparent to the user.

The design features of CODA as providing a complete and progressive design environment can be observed from Figure 1.1 and 1.2.

CODA is built upon ISI's existing CACE product line, i.e., Xmath, SystemBuild, AutoCode, ISIM and RealSim. However, it should be clear from Figure 1.1 and 1.2, but still important to mention, that CODA is not a mere rearrangement of existing

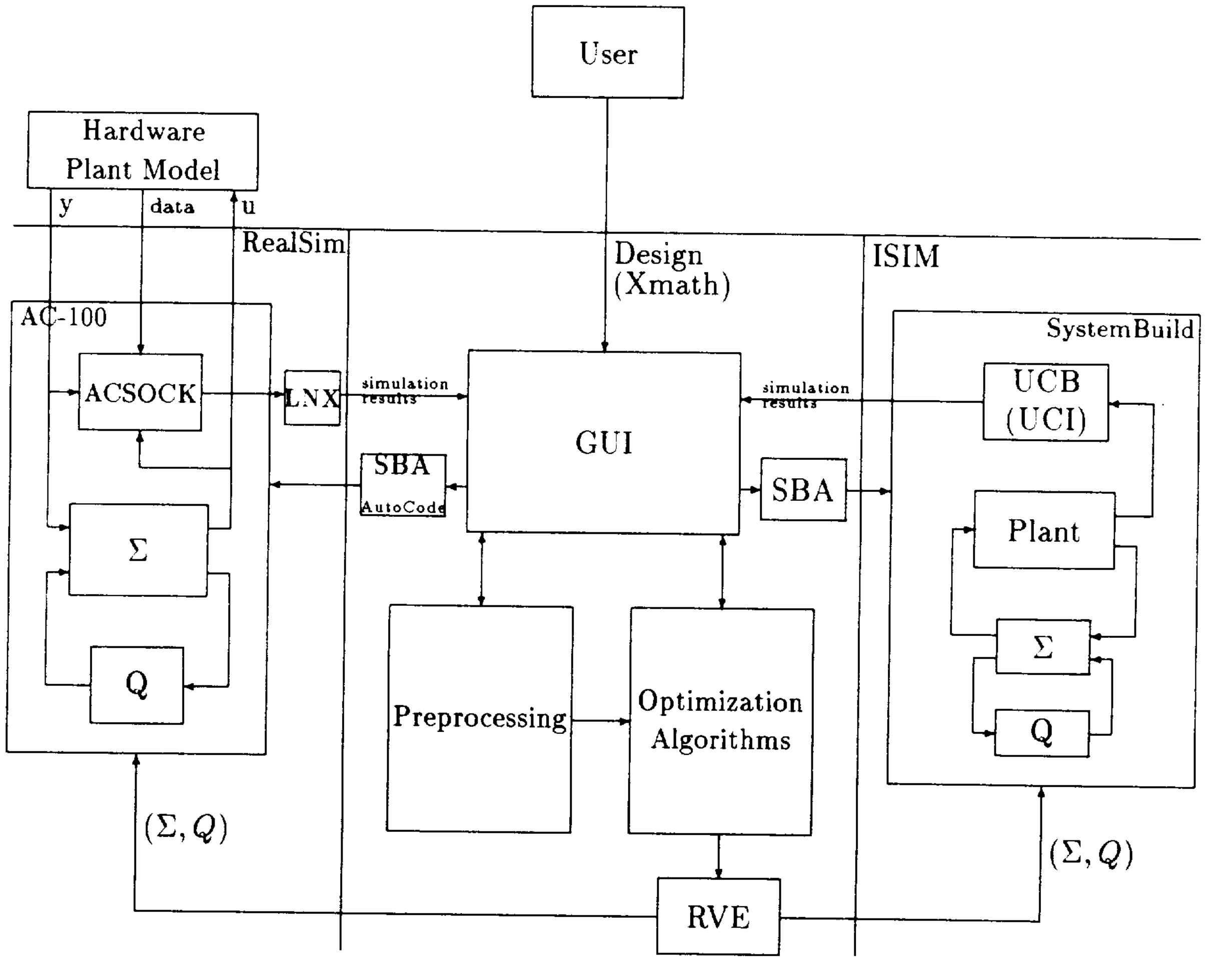


Figure 1.1: Basic CODA architecture

CACE products. What makes CODA completely different than what was possible before, are capabilities that:

- reduce the user-load in translating design problems into various design-tool specific inputs, by supporting a GUI which directly manipulates multiple closed-loop specifications,
- rapidly handle repetitive designs iterations for realistic problem sizes,
- provide efficient optimization algorithms to find a controller design satisfying all of the design specifications,
- enable a seamless interaction with the numerical and the real-time simulation environments (this also reduces the user-load in repetitive numerical and real-time simulation setup and implementation tasks), by allowing the automated implementation of the simulation model, and on-line data processing and controller modification during simulation.

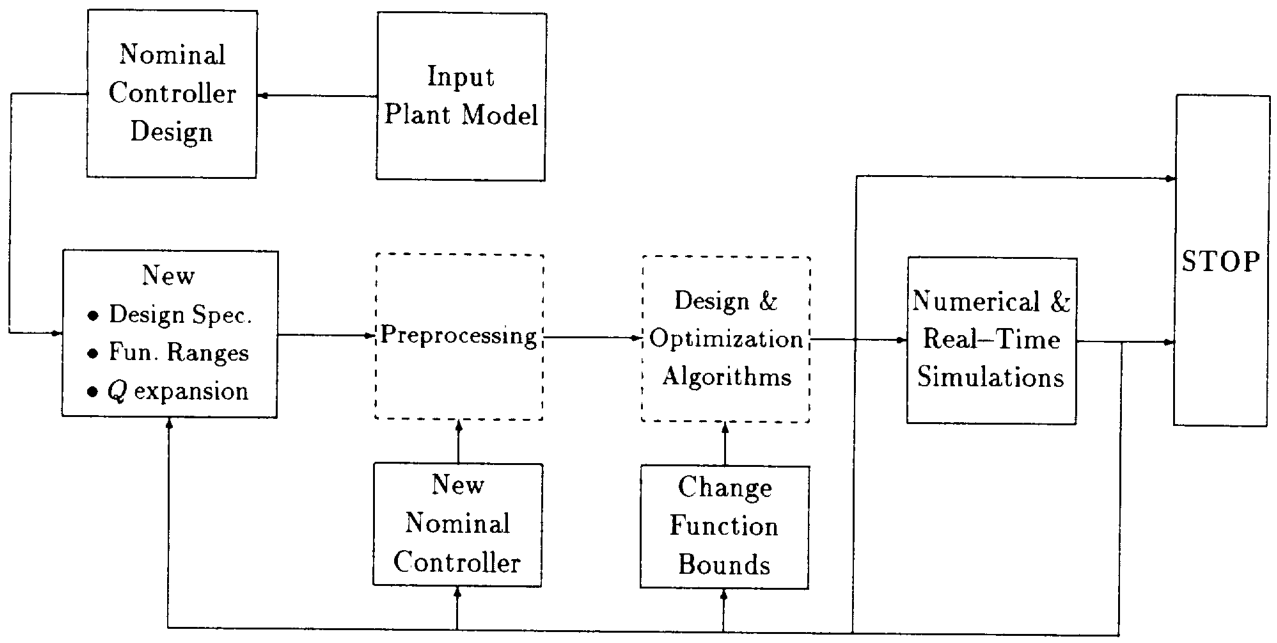


Figure 1.2: A typical design cycle in CODA

The report is outlined in the next section along with a brief description about the functionalities embedded in CODA.

1.3 Outline of Report

It can be observed from Figure 1.1 that CODA consists of the following primary functionalities:

1. Preprocessing scheme
2. Optimization algorithms
3. On-line numerical and real-time simulations
4. GUI

The preprocessing scheme as well as the design approaches and principles of CODA are introduced in Section 2, which include

- Q parametrization for the controller;
- design specifications available in CODA;

- approximation of the Q parameter by a finite-series expansion to reduce the infinite dimensional design problem to a tractable finite dimensional one;
- preprocessing scheme to transform the design specifications into a mathematical convex programming problem.

The optimization algorithms used to solve the mathematical convex programming problems derived from the design specifications are introduced in Section 3, which include

- a descent method based on the non-differentiable optimization theory;
- a barrier-function method which is inspired from the interior-point methods developed recently;
- a hybrid method which uses the descent and the barrier function method alternately in the optimization process.

The optimization algorithms play the role as the 'engine' of CODA without which no design solutions will be produced.

CODA can interact with the ISIM (Interactive SIMulation) module and the RealSim (*Real*time *Sim*ulaton) processor by implementing simulation models and updating designs on an automation and on-line basis via two newly developed functionalities in Xmath: RVE (Run-time Variable Editor) and SBA (SystemBuild Access). The RVE allows the user to change %Variables[1] during an ISIM or RealSim process. The SBA feature provides an Xmath interface into the SystemBuild database, which allows the user to create, modify, query, and delete SystemBuild diagrams. In Section 4, these two functionalities are applied for the interaction of CODA with the ISIM module and the RealSim processor: The SBA feature is applied to implement the simulation model in an automated way, while the RVE is used to update the controller model during the numerical and the real-time simulations. A UCI (User Callable Interface) functionality is also introduced, which is implemented as a UCB (User-Code Block) in SystemBuild. It is used to copy the ISIM results into Xmath and then plot them on the interactive function windows to monitor the ISIM process in an effective way. Similarly, for the real-time simulation case, the outputs of a hardware plant can be transmitted via an ACSOCK ('AC-100 SOCKet' which accesses the TCP/IP communication interface in AC-100 communication (CO) processor) to an LNX (pronounced as 'links') facility through which the data can be copied into Xmath and plotted on the function windows to monitor the real-time simulation process[2]. With these features built in, CODA becomes a CACE environment integrating the multi-objective control system design, implementation of controllers, and simulations together.

[1]%Variables are selected parameters in SystemBuild intrinsic blocks, whose values can be adjusted at simulation time.

[2]As of May 1996, this part remains to be finished.

In Section 5, the GUI facility of CODA is introduced. CODA provides a thorough treatment for multi-objective designs and on-line simulations for the testing of designs, and hence is a very sophisticated CACE design tool. However, its GUI is structured and designed in such a user-friendly way that

- the user's interaction basically stays at the level of design performance specifications;
- with a clear and systematic layout of the CODA main window, and its easy-to-use help system, even the first-time user can learn how to use CODA for design in a minimal time.

1.4 Current ISI CACE Environment

As mentioned before, CODA is built upon ISI's existing CACE product line including Xmath, SystemBuild, AutoCode, ISIM and RealSim. In this section, we give a brief description on ISI's existing CACE design environment.

Xmath is an interactive mathematical analysis and scripting environment, which is written in C++ and is an object-oriented numerical computing tool. It contains a fully programmable GUI design environment which allows windows to be created and manipulated using only Xmath source code (MathScript). Xmath also includes the newly developed functionalities of RVE (Run-time Variable Editor) and SBA (SystemBuild Access) which can be performed within MathScripts via a collection of Xmath commands.

SystemBuild is a model editor and simulator which uses a block diagram GUI to construct linear or nonlinear dynamical systems. The block diagram description is automatically compiled into efficient code for simulation or real-time control purposes. An ISIM process can be invoked in the Xmath command area or within MathScripts to allow the user to change the %Variables in the various blocks of SystemBuild.

The RealSim processor provides automated development of real-time systems from a user supplied block diagram of the controller written in the SystemBuild language. It uses one of the serial processors of 80386, 80486 and 80860, and has animated panels to monitor real-time operation as well some capability for configurable interface design. A PC based version, the RealSim Model C30/C40, is also available and supports a wide variety of PC and TMS320C30/TMS320C40 TI DSP and I/O boards.

The key to the RealSim processor and other real-time applications is ISI's AutoCode which automatically generates C, ADA, or FORTRAN code which can be used on the host or any other processor. The code is generated directly from the SystemBuild block diagrams created by the user. AutoCode is essentially a source code generator that bridges the gap between design and implementation by generating the downloadable code directly from the design database.

2 Design Approaches and Preprocessing Scheme

2.1 4-Block Plant Design Model

As mentioned in Section 1.1, a multi-objective controller design imposes many design requirements such as

- stability
- robustness against unmodeled sector-bounded perturbations
- minimum and maximum bandwidth for the controlled system
- specifications of the rise time and settling time for the step response
- maximum instantaneous amplitudes for each of the plant inputs
- maximum RMS (root-mean-square) levels to noise inputs

and so on. Consider a linear (continuous or discrete) plant model P_0. The first step in a control design is to obtain the augmented plant P. The particular way the augmentation is done depends on the design specifications at hand as well as the adopted design scheme. The goal is to embed P_0 in P so that *the design specifications can be expressed in terms of the closed-loop maps from w to z* in Figure 2.1, where

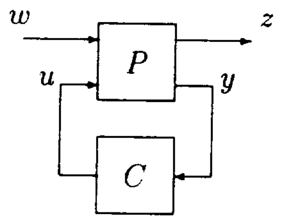


Figure 2.1: Stable interconnection $\mathcal{S}(P, C)$

C denotes a stabilizing controller, and the augmented plant P maps (w, u) to (z, y) with that

- w is the vector of exogenous inputs, typically consisting of all references, disturbances, and noise sources.
- z denotes the regulated outputs consisting of all the variables of interest for design, e.g., states, sensor outputs, actuator inputs, error signals, and so on.
- the vector variables u and y are the actuator inputs and sensor outputs.

Let n_w, n_z, n_u and n_y denote the dimensions of w, z, u and y respectively. It is assumed that all of the undesired modes of P are stabilizable and detectable from the input/output pair (u, y).

Stability is the most basic and important design requirement, and is usually not a trivial design problem especially when the controller is restricted to a certain category, e.g, fixed-order or PID controllers. However, no restrictions on the controller structure are imposed in CODA except the possible requirement of strict properness. In this case, it is known that the set of stabilizing controllers can be characterized by a stable transfer matrix which is usually called *Youla parameter* and denoted by Q [15]. Thus the problem of designing a suitable controller is equivalent to that of finding a suitable Youla parameter Q. By parametrizing the stabilizing controller in terms of the Youla parameter Q, the transfer function from w to z in Figure 2.1 will be affine in Q (see (2.4)). Furthermore, the design requirements in CODA are expressible by convex functions of the closed-loop map from w to z (see Section 2.4). The controller synthesis problem then can be transcribed to a *convex* optimization problem.

The Q parametrization design approach is adopted in CODA. The detail about Q parametrization and related design approaches will be given in the following sections.

2.2 Q (Youla) parametrization

Using possibly a subset of w, z and y, design a full-order estimated-state feedback controller $C : y \rightarrow u$, such that the interconnection $\mathcal{S}(P, C)$ (see Figure 2.1) is stable. This nominal design is used to obtain a parametrization of all stabilizing controllers. The required information is a state-feedback and an output-injection gains. The method of obtaining these gains is immaterial for parametrization; e.g., for scalar u to y maps, one might even use pole-placement. However, from the design point of view, it is advisable to choose the nominal design to meet a subset of the performance specifications for which closed form solutions are available. Let the map from u to y have a state-space description of the form (A, B_u, C_y, D_{yu}). Since the pairs (A, B_u) and (A, C_y) are stabilizable and detectable, respectively, there exist a state-feedback gain K and an output-injection gain L such that $(A - B_u K)$ and $(A - LC_y)$ are stable. Typically, these gains are obtained by solving Riccati equations formulating $\mathcal{H}_2$-optimal or $\mathcal{H}_\infty$-suboptimal designs.

Once such K and L are determined, one obtains a nominal stabilizing controller $C(0) : y \rightarrow u$, with the state-space description $(A - B_u K - LC_y + LD_{yu} K, L, -K, 0)$. The set of all stabilizing controllers from y to u can be expressed in terms of a stable parameter Q, where $C(Q)$ is obtained by the interconnection in Figure 2.2. The map Σ from (y, v) to (u, e) in Figure 2.2 has the following system matrix description[3]:

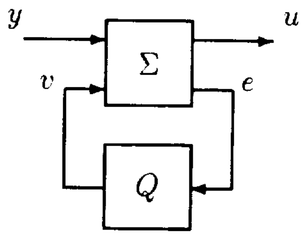


Figure 2.2: All stabilizing controllers $C(Q) : y \rightarrow u$

$$\left[\begin{array}{c|cc} (A - B_uK - LC_y + LD_{yu}K) & L & (B_u - LD_{yu}) \\ \hline -K & 0 & I \\ (D_{yu}K - C_y) & I & -D_{yu} \end{array}\right] . \tag{2.1}$$

Note that v and e have the same dimensions as u and y respectively. For the description above, $C(\cdot) : Q \rightarrow C(Q)$, is one-to-one and onto stabilizing controllers. That is, given any stabilizing controller $\hat{C}$ from y to u, there exists a unique stable transfer function Q such that $C(Q) = \hat{C}$.

With the controller structure given in Figure 2.2, the closed-loop diagram is shown in Figure 2.3. The nominal closed-loop map (with $Q = 0$) from (w, v) to (z, e) can

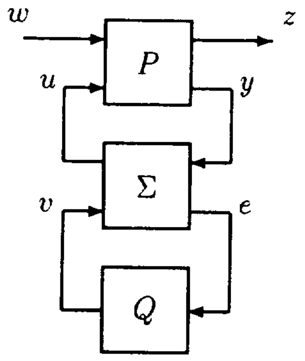


Figure 2.3: All achievable closed-loop maps

be expressed by

$$\begin{bmatrix} z \\ e \end{bmatrix} = \begin{bmatrix} T_{zw} & T_{zv} \\ T_{ew} & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix}, \tag{2.2}$$

[3] $\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$ denotes the dynamical system with

$$\begin{aligned} z &= Ax + Bu, \\ y &= Cx + Du, \end{aligned}$$

where

$$\begin{aligned} z &= \frac{dx}{dt}, \text{ for continuous–time case,} \\ &= x(t+1), \text{ for discrete–time case.} \end{aligned}$$

i.e., the map from v to e is zero. The transfer functions T_{zw}, T_{zv} and T_{ew} in (2.2) are determined by the plant model P and the gains K and L. Define

$$T \triangleq \begin{bmatrix} T_{zw} & T_{zv} \\ T_{ew} & 0 \end{bmatrix}. \tag{2.3}$$

Figure 2.3 is then equivalent to Figure 2.4. It is straightforward to show that the

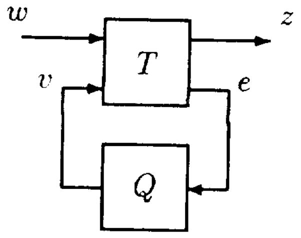


Figure 2.4: A simpler model for all achievable closed-loop maps $H_{zw}(Q)$

closed-loop map $H_{zw}(Q)$ is equal to

$$H_{zw}(Q) \;=\; T_{zw} + T_{zv}\,Q\,T_{ew}, \tag{2.4}$$

which is affine in Q.

In summary, the Q-parametrization design approach consists of two design steps:

1. nominal design

2. Q-parameter design

The nominal design is determined by the state-feedback gain (K) and the output-injection gain (L), and uniquely defines the transfer functions (T_{zv}, T_{zv}, T_{ew}) given in (2.3) by which the set of all closed-loop stable transfer functions is expressed in (2.4) with $Q(\cdot)$ being a stable transfer function. It is clear from (2.4) that T_{zw} is the nominal closed-loop transfer function. Intuitively, a nominal design meeting as much as the design requirements is desirable because it is believed that a 'good' nominal design will make the Q-parameter design easier from the numerical point of view, even though, given any nominal design, theoretically we can always find a Q-parameter to achieve the desired closed-loop transfer function. However, CODA emphasizes the Q-parameter design, even though several options are indeed provided for the nominal controller design in CODA (see Section 5.2.3). The user is referred to ICDM (Interactive Control Design Module) in MATRIX$_X$for a more extensive treatment of the nominal design.

Since the Q parameter belongs to the set of stable matrices which has an infinite dimension, the controller synthesis problem with various design requirements expressed in terms of the closed-loop maps in (2.4) will result in an infinite-dimensional programming problem. A practical approach to solving the infinite-dimensional programming problem is to approximate the design problem by a finite-dimensional one. In the next section, two sets of orthonormal basis functions for the Q parameter are introduced, from which finite-dimensional design problems can be derived.

2.3 Finite-Dimensional Expansion for Q parameter

To transform the general controller synthesis problem with the Q parametrization approach into a finite-dimensional programming problem, the following approximation expansion for the Q parameter is introduced:

$$Q(p,\zeta) = \sum_{k=0}^{n_q} Q_k(p) d_k(\zeta), \qquad (2.5)$$

where $\zeta = s$ or z, n_q is the order of expansion, $\{d_k(\zeta)\}_{k=0}^{n_q}$ denotes the set of expansion basis functions, and p is the design parameter consisting of the elements of the coefficient matrices $\{Q_k\}_{k=0}^{n_q}$. Let n_p be the dimension of the design parameter p. It follows from (2.5) that

$$n_p = (n_q + 1) \cdot n_u \cdot n_y \, .$$

In the Q expansion of (2.5), the variable symbol '$\cdot$' in $d_k(\cdot)$ and $Q(p,\cdot)$ denotes 's', the Laplace parameter, for the continuous-time case, and 'z', the z-transform parameter, for the discrete-time case. It is clear that, given a nominal controller and a set of basis functions, the larger is n_q, the bigger is the dimension of the design parameter, and the closer is the sub-optimal solution to the optimal one.

There are many choices for the expansion basis functions $\{d_k(\cdot)\}$. For example, in [1], $d_k(\cdot)$ is chosen to be

$$d_k(z) = z^{-k} \qquad k \geq 0,$$

for the discrete-time system such that the Q parameter is approximated by an FIR (finite-impulse-response) transfer matrix. However, the set of basis functions $\{z^{-k}\}$ does not have the desired property of orthonormality. In CODA, the Laguerre and the Kautz functions are adopted as two sets of orthonormal basis functions for the Q expansion [13, 14], which are defined as follows:

- **Laguerre Functions**

  Continuous-Time Case:

$$d_k(s) = \frac{\sqrt{2a}}{s+a}\left(\frac{s-a}{s+a}\right)^{k-1}, \quad k \geq 1, \qquad (2.6)$$

  where $a > 0$.

Discrete-Time Case:

$$d_k(z) = \frac{\sqrt{1-a^2}}{z-a}\left(\frac{1-az}{z-a}\right)^{k-1}, \quad k \geq 1, \tag{2.7}$$

where $-1 < a < 1$.

- **Kautz Functions**

Continuous-Time Case:

$$\begin{aligned} d_{2k-1}(s) &= \frac{\sqrt{2bc}}{s^2+bs+c}\left(\frac{s^2-bs+c}{s^2+bs+c}\right)^{k-1}, \\ d_{2k}(s) &= \frac{\sqrt{2c}s}{s^2+bs+c}\left(\frac{s^2-bs+c}{s^2+bs+c}\right)^{k-1}, \end{aligned} \tag{2.8}$$

where $b > 0$, $c > 0$, and $k = 1, 2, \cdots$.

Discrete-Time Case:

$$\begin{aligned} d_{2k-1}(z) &= \frac{\sqrt{1-c^2}(z-b)}{z^2+b(c-1)z-c}\left(\frac{-cz^2+b(c-1)z+1}{z^2+b(c-1)z-c}\right)^{k-1}, \\ d_{2k}(z) &= \frac{\sqrt{(1-c^2)(1-b^2)}}{z^2+b(c-1)z-c}\left(\frac{-cz^2+b(c-1)z+1}{z^2+b(c-1)z-c}\right)^{k-1}, \end{aligned} \tag{2.9}$$

where $-1 < b < 1$, $-1 < c < 1$, and $k = 1, 2, \cdots$.

It is obvious that the Laguerre functions are characterized by a real pole while the Kautz functions are by a pair of complex poles. For the static part of expansion, we choose

$$d_0(\cdot) \equiv 1.$$

If the controller is chosen to be strictly proper, the static part of the Q parameter has to be a zero matrix. In this case, we set

$$Q_0 \equiv \mathbf{0},$$

where $\mathbf{0}$ denotes an $n_u \times n_y$ zero matrix. The dimension of the design parameter, n_p, then becomes

$$n_p = n_q \cdot n_u \cdot n_y .$$

2.4 Design Specifications

In CODA, design specifications can be imposed on the following functions or responses:

$\mathcal{H}_2$ (RMS) functions on which upper bounds are assigned to, e.g., set up the maximum RMS levels to disturbance/noise inputs.

Step responses on which upper and lower bounds are specified to put constraints on rising times, settling times, overshoot, asymptotic values, etc.

Impulse responses on which upper and lower bounds are assigned to impose constraints on the amplitudes of an impulse response, e.g., from a reference input to a sensor output.

MSV (maximum-singular-value) responses in frequency domain, on which upper bounds are specified to put constraints on gains, bandwidths and rolling rates to achieve the design goals like robustness to unmodeled perturbations, disturbance/noise rejections, bandwidth assignments, etc.

Beside the design specifications on the above 4 types of functions and responses, the tracking requirement among selected input-output pairs is very common in control system design. Therefore:

Integral actions can be imposed on the selected input-output pairs so that the outputs will achieve the asymptotic values assigned by the user for the step inputs, i.e., zero steady-state errors are guaranteed.

2.5 Problem Formulation

With the Q parameter approximated by a finite-dimensional expansion as shown in (2.5), each of the design requirements considered in Section 2.4 can be formulated by a convex function.

2.5.1 $\mathcal{H}_2$ Design Function

The $\mathcal{H}_2$ or RMS function, $F(p)$, for a given input-output pair (w_1, z_1) with $w_1 \subset w$ and $z_1 \subset z$, is defined to be

$$\begin{aligned} F^2(p) &= 2 \cdot tr \left(\int_0^\infty H^*_{z_1 w_1}(p, jf) H_{z_1 w_1}(p, jf)\, df \right), \quad \text{for continuous–time case} \\ &= 2T_0 \cdot tr \left(\int_0^{\frac{1}{2T_0}} H^*_{z_1 w_1}(p, e^{j2\pi T_0 f}) H_{z_1 w_1}(p, e^{j2\pi T_0 f})\, df \right), \\ &\qquad \text{for discrete–time case,} \end{aligned} \tag{2.10}$$

where $tr(M)$ denotes the trace (the sum of diagonal elements) of the matrix M, T_0 is the period of the discrete-time system, H^* denotes the complex conjugate transpose of a complex matrix H, and

$$H_{z_1 w_1}(p,\cdot) \;=\; T_{z_1 w_1}(\cdot) + \sum_{k=0}^{n_q} d_k(\cdot) T_{z_1 v}(\cdot) Q_k(p) T_{e w_1}(\cdot). \tag{2.11}$$

It is clear from (2.10) that the $\mathcal{H}_2$ function $F(\cdot)$ can be expressed by

$$F(p) \;=\; \sqrt{\frac{1}{2} p^T H p + p^T g + f_0}, \tag{2.12}$$

where H is a non-negative definite Hessian matrix, g is a gradient vector and f_0 is a scalar. Suppose $b_1 > 0$ is the upper bound for the $\mathcal{H}_2$ function. The RMS constraint can then be expressed by

$$F(p) \le b_1. \tag{2.13}$$

Define the $\mathcal{H}_2$ design function $\phi(\cdot)$ to be

$$\begin{aligned} \phi(p) \;&\stackrel{\triangle}{=}\; F(p) - b_1 \\ &=\; \sqrt{\frac{1}{2} p^T H p + p^T g + f_0} - b_1, \end{aligned} \tag{2.14}$$

which is the deviation of the RMS value $F(p)$ from the upper bound b_1. Since $F(\cdot)$ is a convex function, so is $\phi(\cdot)$. Equation (2.13) is then equivalent to

$$\phi(p) \le 0. \tag{2.15}$$

2.5.2 Step/Impulse Design Functions

Consider a step or impulse response from the input $w_2 \in w$ to the output $z_2 \in z$. Referring to (2.4) and (2.5), the closed-loop transfer function from w_2 to z_2 is equal to

$$H_{z_2 w_2}(p,\cdot) \;=\; T_{z_2 w_2}(\cdot) + \sum_{k=0}^{n_q} d_k(\cdot) T_{z_2 v}(\cdot) Q_k(p) T_{e w_2}(\cdot). \tag{2.16}$$

It is straightforward to show that each step or impulse responses corresponding to n_s equally spaced sampling points can be represented by

$$A \cdot p + b, \tag{2.17}$$

where A is an $n_s \times n_p$ matrix, and b is an $n_s \times 1$ vector. Let $\bar{b}$ and $\underline{b}$, which are $n_s \times 1$ vectors, denote the upper and the lower bounds respectively for the response. The design requirements for step and impulse responses can be formulated by

$$\underline{b} \le A \cdot p + b \le \bar{b}, \tag{2.18}$$

which is equivalent to

$$\begin{bmatrix} A \\ -A \end{bmatrix} p + \begin{bmatrix} b - \bar{b} \\ \underline{b} - b \end{bmatrix} \le 0. \tag{2.19}$$

Define

$$\tilde{A} \triangleq \begin{bmatrix} A \\ -A \end{bmatrix}, \qquad \tilde{b} \triangleq \begin{bmatrix} b - \bar{b} \\ \underline{b} - b \end{bmatrix}.$$

Equation (2.19) can then be rewritten as

$$\tilde{A} \cdot p + \tilde{b} \leq 0, \tag{2.20}$$

Define the step or impulse design function, $\phi(p)$, to be

$$\phi(p) \triangleq \max(\tilde{A}p + \tilde{b}), \tag{2.21}$$

which is equal to the maximum deviation of the response to the bounds. It is easy to show that $\phi(\cdot)$ defined in (2.21) is a convex function. Clearly (2.19) is equivalent to

$$\phi(p) \leq 0. \tag{2.22}$$

2.5.3 MSV Design Function

Consider an MSV response from the input $w_3 \subset w$ to the output $z_3 \subset z$ with the closed-loop transfer function:

$$H_{z_3 w_3}(p, \cdot) = T_{z_3 w_3}(\cdot) + \sum_{k=0}^{n_q} d_k(\cdot) T_{z_3 v}(\cdot) Q_k(p) T_{e w_3}(\cdot). \tag{2.23}$$

The MSV response, $h(p, f)$, is defined to be

$$\begin{aligned} h(p,f) &\triangleq \|H_{z_3 w_3}(p, j2\pi f)\|, \text{ for continuous–time case,} \\ &\triangleq \|H_{z_3 w_3}(p, e^{j2\pi T_0 f})\|, \text{ for discrete–time case,} \end{aligned}$$

where T_0 is the period of the discrete-time system, and $\|H\|$ denotes the induced 2-norm of the complex matrix H. It is known that $\|H\|$ is equal to the maximum singular value of H. Since $H_{z_3 w_3}(p, \cdot)$ is affine in p, it is straightforward to show that $h(p, f)$ is a convex function of p. Suppose $\bar{b}(\cdot)$ is the upper bound for the MSV response, and $\Omega \triangleq \{f_i\}_{i=1}^{n_f}$ denotes the frequency sampling points. The MSV constraints can then be expressed by

$$h(p, f) - \bar{b}(f) \leq 0 \qquad \forall f \in \Omega. \tag{2.24}$$

Define the convex MSV design function, $\phi(p)$, to be

$$\phi(p) \triangleq \max_{f \in \Omega} \left(h(p, f) - \bar{b}(f) \right), \tag{2.25}$$

which is equal to the maximum deviation of the MSV response to the upper bound $\bar{b}(\cdot)$. Equation (2.24) is then equivalent to

$$\phi(p) \leq 0. \tag{2.26}$$

2.5.4 Integral-Action Requirements

Suppose integral action is required from w_0 to z_0 with $w_0 \subset w$, $z_0 \subset z$ and

$$\dim(w_0) = \dim(z_0) = n_0.$$

It will impose the following linear equality constraints on the design parameter p:

$$T_{z_0 w_0}(0) + \sum_{k=0}^{n_q} d_k(0) T_{z_0 v}(0) Q_k(p) T_{e w_0}(0) \;=\; \mathrm{diag}(\gamma_1, \gamma_2, \dots, \gamma_{n_0}), \qquad (2.27)$$

for continuous–timecase

$$T_{z_0 w_0}(1) + \sum_{k=0}^{n_q} d_k(1) T_{z_0 v}(1) Q_k(p) T_{e w_0}(1) \;=\; \mathrm{diag}(\gamma_1, \gamma_2, \dots, \gamma_{n_0}), \qquad (2.28)$$

for discrete–timecase

where γ_i, $1 \le i \le n_0$, is the desired asymptotic value of the step response for the i-th tracking pair. It is clear that (2.27) and (2.28) impose linear inequality constraints on the design parameter p. Hence the design parameter p can be expressed by

$$p \;=\; A_0 \tilde{p} + b_0, \qquad (2.29)$$

where A_0 is an $n_p \times n_{\tilde{p}}$ matrix and b_0 is an $n_p \times 1$ vector with $n_{\tilde{p}} \le n_p$. Suppose there exist no poles nor zeros at $s = 0$ (for the continuous-time case) or $z = 1$ (for the discrete-time case) in the plant model for each tracking channel. In this case, we have

$$n_{\tilde{p}} \;=\; n_p - n_0^2.$$

which implies that the dimension of the design parameter n_p should be larger than n_0^2 in this case.

2.5.5 Conclusion

In summary, with the design specifications considered in Section 2.4 and the Q parameter approximated by (2.5), the design problem can be transformed into the following convex feasibility programming problem:

$$\mathbf{OP}: \quad \begin{cases} \phi_1(p) & \le 0 \\ \phi_2(p) & \le 0 \\ \quad\vdots & \\ \phi_m(p) & \le 0 \end{cases} \qquad (2.30)$$

where m is the total number of design requirements, and each ϕ_i, $1 \le i \le m$, is in the form of (2.15), (2.20) or (2.26). If there exist tracking requirements, the design parameter space will be reduced from p to $\tilde{p}$ according to (2.29). Since there exists an affine relationship between p and $\tilde{p}$, the feasibility programming problem (2.30) remains to be a convex one.

A nice property about the convex feasibility problem in (2.30) is that the set of feasible solutions is either an empty or a convex set. That is, if p_1 and p_2 are two solution points of the problem (**OP**), a convex combination

$$\lambda p_1 + (1-\lambda)p_2 \qquad \text{with} \qquad 0 \leq \lambda \leq 1,$$

is also a feasible solution due to the fact that each $\phi_i(\cdot)$, $1 \leq i \leq m$, is a convex function and, therefore

$$\begin{aligned}\phi_i(\lambda p_1 + (1-\lambda)p_2) &\leq \lambda\phi_i(p_1) + (1-\lambda)\phi_i(p_2)\\ &\leq 0.\end{aligned}$$

Another nice property about the convex programming problem is that the local minima problems do not exist.

2.6 Preprocessing Scheme

With a given nominal design, the preprocessing scheme in CODA is used to acquire the following data set for any Q expansion:

- $(H,\ g,\ f_0)$ for the $\mathcal{H}_2$ function (see (2.12));
- $(A,\ b)$ for the step or impulse response (see (2.17));
- the frequency responses of $(T_{z_3w_3}(\cdot),\ T_{z_3v}(\cdot),\ T_{ew_3}(\cdot))$ and $\{d_k(\cdot)\}_{k=1}^{n_q}$ for the MSV response (see (2.23));
- $(A_0,\ b_0)$ for the integral-action constraint (see (2.29)).

These data sets are needed in finding a feasible solution for the programming problem (**OP**).

The acquisition of the frequency responses for the MSV response and $(A_0,\ b_0)$ for the integral-action constraint from (2.27) or (2.28) is pretty straightforward. Therefore, we will focus on the preprocessing methodologies for the $\mathcal{H}_2$ function and the step/impulse response in this section.

In the implementation of the preprocessing scheme, we should take account of the fact that the preprocessing scheme is invoked each time the user changes the Q expansion. Therefore it should be kept in mind that the preprocessing scheme to be implemented be efficient for frequent preprocessing calls.

Note that the definitions and notations given in Section 2.5 will be borrowed here.

2.6.1 Preprocessing for $\mathcal{H}_2$ Function

The preprocessing scheme for the $\mathcal{H}_2$ function is aimed at obtaining the data set (H, g, f_0) in (2.12). An exact state-space-based approach requires solving $(n_p + 2)$ Riccati equations, which takes so much CPU time for the design problem with a reasonable plant order (say, ≥ 30) and parameter dimension (say, ≥ 40), that it becomes unacceptable [9]. Therefore an approximation preprocessing is adopted for the $\mathcal{H}_2$ function in CODA. Let's consider the continuous-time case first.

Let p_i denote the (α_i, β_i)-th element of Q_k with $1 \leq \alpha_i \leq n_u$, $1 \leq \beta_i \leq n_y$ and $0 \leq k \leq n_q$. Note that (2.11) can be rewritten as

$$H_{z_1 w_1}(s) = T_0(s) + \sum_{i=1}^{n_p} T_i(s) p_i, \tag{2.31}$$

where

$$T_0(\cdot) \triangleq T_{z_1 w_1}(\cdot),$$

and

$$T_i(s) = T_{z_1 v}^{\cdot,\alpha_i}(s) \cdot T_{e w_1}^{\beta_i,\cdot}(s) \cdot d_k(s).$$

Note that $T_{z_1 v}^{\cdot,\alpha_i}(\cdot)$ is the α_i-th column of $T_{z_1 v}(\cdot)$, and $T_{e w_1}^{\beta_i,\cdot}(\cdot)$ is the β_i-th row of $T_{e w_1}(\cdot)$. From the definition of the $\mathcal{H}_2$-norm (see (2.10)), we have

$$\begin{aligned} F^2(p) &= 2 \cdot \mathrm{tr}\left(\int_0^\infty H_{z_1 w_1}^*(p, j2\pi f) H_{z_1 w_1}(p, j2\pi f)\, df\right) \\ &= \frac{1}{2}\left[4 \sum_{i=1}^{n_p}\sum_{k=1}^{n_p} \mathrm{Re}\left(\int_0^\infty tr(T_i^*(j2\pi f) T_k(j2\pi f)) p_i p_k\, df\right)\right] + \\ & \quad 4 \sum_{i=1}^{n_p} \mathrm{Re}\left(\int_0^\infty tr(T_0^*(j2\pi f) T_i(j2\pi f)) p_i\, df\right) + \\ & \quad 2 \cdot \mathrm{tr}\left(\int_0^\infty T_0^*(j2\pi f) T_0(j2\pi f)\, df\right) \end{aligned} \tag{2.32}$$

where Re$(\cdot)$ denotes the real part of the complex argument. Therefore we have

$$f_0 = 2 \cdot \mathrm{tr}\left(\int_0^\infty T_0^*(j2\pi f) T_0(j2\pi f)\, df\right), \tag{2.33}$$

$$g_i = 4 \cdot \mathrm{Re}\left(\int_0^\infty tr(T_0^*(j2\pi f) T_i(j2\pi f))\, df\right), \tag{2.34}$$

$$h_{ik} = 4 \cdot \mathrm{Re}\left(\int_0^\infty tr(T_i^*(j2\pi f) T_k(j2\pi f))\, df\right), \tag{2.35}$$

where g_i and h_{ik} are, respectively, the i-th and the (i, k)-th element of g and H with $1 \leq i, k \leq n_p$. To have an efficient evaluation of f_0, g and H for various Q expansions, let M_0 be a set of $(n_u \cdot n_y)$ indexs such that $\{p_i\}_{i \in M_0}$ is the set of elements of Q_0 in (2.5). Corresponding to each p_m with $m \in M_0$, let $G_m(\cdot)$ be defined by

$$\begin{aligned} G_m(f) &\triangleq tr(\,T_0^*(j2\pi f) T_m(j2\pi f)\,) = tr(\,T_0^*(j2\pi f) \cdot T_{z_1 v}^{\cdot,\alpha_i}(j2\pi f) \cdot T_{e w_1}^{\beta_i,\cdot}(j2\pi f)\,) \\ &= T_{e w_1}^{\beta_i,\cdot}(j2\pi f)\, T_0^*(j2\pi f)\, T_{z_1 v}^{\cdot,\alpha_i}(j2\pi f). \end{aligned} \tag{2.36}$$

Similarly, let $H_{m_1 m_2}(\cdot)$ with $m_1 \in M_0$ and $m_2 \in M_0$ be defined by

$$\begin{aligned} H_{m_1 m_2}(f) &\triangleq tr(\, T^*_{m_1}(j2\pi f) T_{m_2}(j2\pi f)\,) \\ &= tr(\, \left(T^{\cdot,\alpha_{m_1}}_{z_1 v}(j2\pi f) \cdot T^{\beta_{m_1},\cdot}_{ew_1}(j2\pi f)\right)^* \left(T^{\cdot,\alpha_{m_2}}_{z_1 v}(j2\pi f) \cdot T^{\beta_{m_2},\cdot}_{ew_1}(j2\pi f)\right)\,) \\ &= \left((T^{\cdot,\alpha_{m_1}}_{z_1 v}(j2\pi f))^* T^{\cdot,\alpha_{m_2}}_{z_1 v}(j2\pi f)\right) \left(T^{\beta_{m_2},\cdot}_{ew_1}(j2\pi f)(T^{\beta_{m_1},\cdot}_{ew_1}(j2\pi f))^*\right). \end{aligned} \tag{2.37}$$

Suppose p_i is the (α_i, β_i)-th element of Q_{k_i}. Let p_{m_i}, with $m_i \in M_0$, be the (α_i, β_i)-th element of Q_0. Equation (2.34) can be rewritten as

$$g_i = 4 \cdot \mathrm{Re}\left(\int_0^\infty G_{m_i}(f) d_{k_i}(j2\pi f)\, df\right). \tag{2.38}$$

Suppose p_j is the (α_j, β_j)-th element of Q_{k_j}. Let p_{m_j}, with $m_j \in M_0$, be the (α_j, β_j)-th element of Q_0. It is straightforward to shown that (2.35) is equal to

$$h_{ij} = 4 \cdot \mathrm{Re}\left(\int_0^\infty H_{m_i m_j}(f) d^*_{k_i}(j2\pi f) d_{k_j}(j2\pi f)\, df\right). \tag{2.39}$$

Once $\{G_m(\cdot)\}_{m \in M_0}$, $(T^*_{z_1 v}(\cdot) T_{z_1 v}(\cdot))$, and $\left(T_{ew_1}(\cdot) T^*_{ew_1}(\cdot)\right)^T$ be determined and stored in the data stack, the evaluation of g and H via (2.37), (2.38) and (2.39) will become very efficient.

The discrete-time case is similar to the continuous-time case. Substituting (2.31) into (2.10), we get

$$f_0 = 2T_0 \cdot tr\left(\int_0^{\frac{1}{2T_0}} T_0^*(e^{j2\pi T_0 f}) T_0(e^{j2\pi T_0 f})\, df\right), \tag{2.40}$$

$$g_i = 4T_0 \cdot \mathrm{Re}\left(\int_0^{\frac{1}{2T_0}} tr(T_0^*(e^{j2\pi T_0 f}) T_i(e^{j2\pi T_0 f}))\, df\right), \tag{2.41}$$

$$h_{ij} = 4T_0 \cdot \mathrm{Re}\left(\int_0^{\frac{1}{2T_0}} tr(T_i^*(e^{j2\pi T_0 f}) T_j(e^{j2\pi T_0 f}))\, df\right). \tag{2.42}$$

The evaluation of f_0, g and H for the continuous-time case can then be applied for the discrete-time case with $(j2\pi f)$ replaced by $(e^{j2\pi T_0 f})$ and the integral range changed from $[0,\ \infty)$ to $[0,\ \frac{1}{2T_0}]$.

The preprocessing scheme discussed in this section is just an approximation because a finite sum is used to approximate the integrals in (2.38) and (2.39).

2.6.2 Preprocessing for Step/Impulse Responses

Consider the time (step or impulse) response for the closed-loop map from $w_2 \in w$ to $z_2 \in z$ with the transfer function given in (2.16) which are repeated here:

$$H_{z_2 w_2}(p, \cdot) = T_{z_2 w_2}(\cdot) + \sum_{k=0}^{n_q} d_k(\cdot) T_{z_2 v}(\cdot) Q_k(p) T_{ew_2}(\cdot).$$

The preprocessing for the time response is to find the matrix A and the vector b such that the step or impulse response can be evaluated by (see (2.17)):

$$A \cdot p + b\,.$$

It is clear that the vector b is equal to the time response of the transfer function

$$T_{z_2 w_2}(\cdot).$$

Let

$$p \triangleq [p_0^T \; p_1^T \; \cdots \; p_{n_q}^T]^T,$$

with

$$p_k \triangleq [p_1^{(k)} \; p_2^{(k)} \; \cdots \; p_{n_u \times n_y}^{(k)}]^T,$$

denoting the design parameter which consists of the elements of Q_k. Let A be decomposed by

$$A \triangleq [A_0 \;\; A_1 \;\; \cdots \;\; A_{n_q}],$$

where each A_k, $0 \leq k \leq n_q$, is an $n_s \times (n_u \cdot n_y)$ matrix. Clearly, each column vector of A_k is determined by the time response of an associated element function of

$$d_k(\cdot)\left(T_{z_2 v}(\cdot)^T \cdot T_{e w_2}(\cdot)^T\right). \tag{2.43}$$

In particular, the matrix A_0 is determined by the time response of the $n_u \times n_y$ transfer matrix $\tilde{T}(\cdot)$ defined by

$$\tilde{T}(\cdot) \triangleq T_{z_2 v}(\cdot)^T \cdot T_{e w_2}(\cdot)^T. \tag{2.44}$$

Suppose $T_{z_2 v}(\cdot)$ and $T_{e w_2}(\cdot)$ have the state-space realizations of

$$\left[\begin{array}{c|c} A_t & B_v \\ \hline C_{z_2} & D_{z_2 v} \end{array}\right] \quad \text{and} \quad \left[\begin{array}{c|c} A_t & B_{w_2} \\ \hline C_e & D_{e w_2} \end{array}\right],$$

respectively. The transfer function $\tilde{T}(\cdot)$ in (2.44) then can be realized by

$$\left[\begin{array}{c|c} \tilde{A} & \tilde{B} \\ \hline \tilde{C} & \tilde{D} \end{array}\right] \triangleq \left[\begin{array}{c|c} \begin{pmatrix} A_t^T & 0 \\ C_{z_2}^T B_{w_2}^T & A_t^T \end{pmatrix} & \begin{pmatrix} C_e^T \\ C_{z_2}^T D_{e w_2}^T \end{pmatrix} \\ \hline \begin{bmatrix} D_{z_2 v}^T B_{w_2}^T & B_v^T \end{bmatrix} & D_{z_2 v}^T D_{e w_2}^T \end{array}\right]. \tag{2.45}$$

There exists a simple, and yet efficient, way to evaluate the matrix A for the discrete-time case. Referring to Section 2.3, it is straightforward to show that

$$d_k(z) = \left(\frac{1 - az}{z - a}\right) \cdot d_{k-1}(z), \quad k \geq 1, \tag{2.46}$$

for the Laguerre expansion, and

$$d_k(z) = \left(\frac{-cz^2 + b(c-1)z + 1}{z^2 + b(c-1)z - c}\right) \cdot d_{k-2}(z), \quad k \geq 2, \tag{2.47}$$

for the Kautz expansion. Therefore, we have the following recursive equations: for the case of Laguerre expansion,

$$\begin{aligned} A_1 &= \left(\frac{\sqrt{1-a^2}}{z-a}\right) * A_0, \\ A_k &= \left(\frac{1-az}{z-a}\right) * A_{k-1}, \qquad k \geq 2 \; ; \end{aligned} \tag{2.48}$$

for the case of Kautz expansion,

$$\begin{aligned} A_1 &= \left(\frac{\sqrt{1-c^2}(z-b)}{z^2+b(c-1)z-c}\right) * A_0, \\ A_2 &= \left(\frac{\sqrt{(1-c^2)(1-b^2)}}{z^2+b(c-1)z-c}\right) * A_0, \\ A_k &= \left(\frac{-cz^2+b(c-1)z+1}{z^2+b(c-1)z-c}\right) * A_{k-2}, \qquad k \geq 3 \; ; \end{aligned} \tag{2.49}$$

where '$*$' denotes the convolution operator. Equations (2.48) and (2.49) mean that the i-th column of A_k can be obtained by convolving the i-th column of (i) A_{k-1} with a 1st-order dynamics for the case that Laguerre expansion is used, or (ii) A_{k-2} with a 2nd-order system for the case of Kautz expansion. Therefore the update of A due to the change of poles and/or order of the Q-expansion can be accomplished by the convolution of the current data of A with a 1st or 2nd order system, which makes the preprocessing for the step/impulse responses of discrete-time systems very efficient.

The preprocessing scheme for the continuous-time system is more complicated and has to go back to (2.43). Let

$$\tilde{Q}(s) = \left[d_0(s)I_{n_y} \; d_1(s)I_{n_y} \; d_2(s)I_{n_y} \; \ldots \; d_{n_q}(s)I_{n_y}\right], \qquad \text{if } n_u \geq n_y,$$

or

$$\tilde{Q}(s) = \begin{bmatrix} d_0(s)I_{n_u} \\ d_1(s)I_{n_u} \\ \vdots \\ d_{n_q}(s)I_{n_u} \end{bmatrix}, \qquad \text{if } n_u < n_y,$$

where I_m denotes the identity matrix with order m. It follows from (2.43) that the matrix A can be obtained by evaluating the time (step or impulse) responses of the systems shown in Figures 2.5 and 2.6, where the dimensions of $\tilde{u}_1$, $\tilde{y}_1$, $\tilde{u}_2$ and $\tilde{y}_2$ are $(n_q+1)n_y$, n_u, n_y and $(n_q+1)n_u$ respectively. Suppose $\tilde{Q}(\cdot)$ is represented by the state-space realization

$$\left[\begin{array}{c|c} A_q & B_q \\ \hline C_q & D_q \end{array}\right].$$

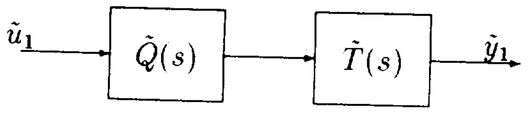


Figure 2.5: The preprocessing model for continuous time responses with $n_u \geq n_y$

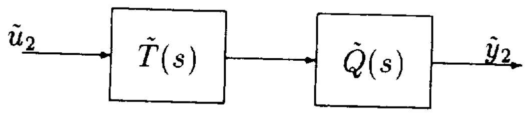


Figure 2.6: The preprocessing model for continuous time responses with $n_u < n_y$

It follows from (2.45) that the cascade systems in Figure 2.5 and Figure 2.6 can be described by

$$\left[\begin{array}{c|c} \begin{pmatrix} \tilde{A} & \tilde{B}C_q \\ 0 & A_q \end{pmatrix} & \begin{pmatrix} \tilde{B}D_q \\ B_q \end{pmatrix} \\ \hline \begin{bmatrix} \tilde{C} & \tilde{D}C_q \end{bmatrix} & \tilde{D}D_q \end{array}\right], \tag{2.50}$$

and

$$\left[\begin{array}{c|c} \begin{pmatrix} \tilde{A} & 0 \\ B_q\tilde{C} & A_q \end{pmatrix} & \begin{pmatrix} \tilde{B} \\ B_q\tilde{D} \end{pmatrix} \\ \hline \begin{bmatrix} D_q\tilde{C} & C_q \end{bmatrix} & D_q\tilde{D} \end{array}\right], \tag{2.51}$$

respectively.

The preprocessing scheme based on Figures 2.5 and 2.6 and, hence, (2.50) and (2.51) can be simplified if $\tilde{T}(\cdot)$ and $\tilde{Q}(\cdot)$ do not have common poles. For this case, consider the state transformation matrices

$$\begin{pmatrix} I_{n_0} & T_1 \\ 0 & I_{ns_q} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} I_{n_0} & 0 \\ T_2 & I_{ns_q} \end{pmatrix}$$

relating the new state coordinate to the old state coordinate for the systems in (2.50) and (2.51) respectively, with T_1 and T_2 satisfying

$$\tilde{A}T_1 - T_1A_q = -\tilde{B}C_q, \tag{2.52}$$

and

$$A_qT_2 - T_2\tilde{A} = -B_q\tilde{C}. \tag{2.53}$$

Note that n_0 and ns_q are respectively the orders of systems $\tilde{T}(\cdot)$ and $\tilde{Q}(\cdot)$, and T_1 and T_2 are $n_0 \times ns_q$ and $ns_q \times n_0$ matrices respectively. The systems in (2.50) and (2.51) are then equivalent to

$$\left[\begin{array}{c|c} \begin{pmatrix} \tilde{A} & 0 \\ 0 & A_q \end{pmatrix} & \begin{pmatrix} \tilde{B}D_q - T_1B_q \\ B_q \end{pmatrix} \\ \hline \begin{bmatrix} \tilde{C} & \tilde{C}T_1 + \tilde{D}C_q \end{bmatrix} & \tilde{D}D_q \end{array}\right], \tag{2.54}$$

and

$$\left[\begin{array}{c|c} \begin{pmatrix} \tilde{A} & 0 \\ 0 & A_q \end{pmatrix} & \begin{pmatrix} \tilde{B} \\ -T_2\tilde{B}+B_q\tilde{D} \end{pmatrix} \\ \hline \begin{bmatrix} D_q\tilde{C}+C_qT_2 & C_q \end{bmatrix} & D_q\tilde{D} \end{array}\right], \tag{2.55}$$

respectively. We can evaluate and store the time (step or impulse) responses of

$$\left[\begin{array}{c|c} \tilde{A} & I_{n_0} \\ \hline \tilde{C} & 0 \end{array}\right],$$

denoted by $Y_1(\cdot)$, for the system in (2.54), and the time responses of

$$\left[\begin{array}{c|c} \tilde{A} & \tilde{B} \\ \hline I_{n_0} & 0 \end{array}\right],$$

denoted by $Y_2(\cdot)$, for the system in (2.55). The time responses of the system in (2.54) is then equal to

$$Y_1 \cdot (\tilde{B}D_q - T_1B_q) + \text{Resp}\left(\left[\begin{array}{c|c} A_q & B_q \\ \hline \tilde{C}T_1+\tilde{D}C_q & \tilde{D}D_q \end{array}\right]\right), \tag{2.56}$$

and the time responses of the system in (2.55) is equal to

$$(D_q\tilde{C} + C_qT_2) \cdot Y_2 + \text{Resp}\left(\left[\begin{array}{c|c} A_q & -T_2\tilde{B}+B_qD_0 \\ \hline C_q & D_qD_0 \end{array}\right]\right), \tag{2.57}$$

where Resp(S) indicates the time response of a system S.

2.6.3 Special Case with Zero K and L

Suppose the plant model is stable, and the state feedback gain (K) and the output injection gain (L) are chosen to be zeros. In this special case, the preprocessing for all of the functions and responses can be simplified due to the fact that $T_{zw}(\cdot)$, $T_{zv}(\cdot)$ and $T_{ew}(\cdot)$ in (2.3) are reduced to

$$T_{zw}(\cdot) \rightarrow P_{zw}(\cdot), \qquad T_{zv}(\cdot) \rightarrow P_{zu}(\cdot), \qquad T_{ew}(\cdot) \rightarrow P_{yw}(\cdot),$$

where $P_{zw}(\cdot)$, $P_{zu}(\cdot)$ and $P_{yw}(\cdot)$ are the open-loop transfer functions of the 4-block plant model P in Figure 2.1 from w to z, u to z, and w to y, respectively. Note that the order of the system $(P_{zw}(\cdot), P_{zu}(\cdot), P_{yw}(\cdot))$ is only a half of that of $(T_{zw}(\cdot), T_{zv}(\cdot), T_{ew}(\cdot))$.

2.6.4 Conclusion

The design process in CODA may involve various Q expansions, and the dimension of design parameter space can be very large. The preprocessing methodologies studied in Section 2.6.2 has taken account of this fact and proved by numerical simulations that it results in a very efficient preprocessing scheme for CODA. The preprocessing scheme will be invoked each time whenever

- a new design specification is defined;
- the sampling points of step/impulse or MSV responses change.

The preprocessing scheme will also be invoked for the design specifications already defined if

- a new nominal controller is adopted;
- a new Q expansion is used.

After the preprocessing is done, the existing design requirements should have been transformed into the optimization problem (**OP**) in (2.30) which is ready for the user to invoke the optimization solvers in CODA to find a feasible solution.

3 Optimization Algorithms

3.1 Introduction

As a summary of Section 2, the control system design problem considered in CODA can be transformed into a convex feasibility programming problem (see (2.30)):

$$\mathbf{OP}: \qquad \begin{cases} \phi_1(p) & \leq 0 \\ \phi_2(p) & \leq 0 \\ \quad\vdots & \\ \phi_m(p) & \leq 0 \end{cases}$$

where m is the total number of design requirements, and each ϕ_i, $1 \leq i \leq m$, is a convex function and in either form of (see Section 2.5 for definitions of notations)

- $\sqrt{\frac{1}{2}p^T Hp + p^T g + f_0} - b_1$, for $\mathcal{H}_2$(RMS) constraints;
- $\max\left([(Ap + b - \bar{b})^T,\ (\underline{b} - Ap - b)^T]\right)$, for step/impulse constraints;
- $\max_{f_i \in \Omega}\left(h(p, f) - \bar{b}(f_i)\right)$, for MSV constraints.

If there exist tracking requirements, a new design parameter space $\tilde{p}$ will be derived from the old one p by

$$p \ = \ A_0\tilde{p} + b_0,$$

where A_0 is an $n_p \times n_{\tilde{p}}$ matrix and b_0 is an $n_p \times 1$ vector with $n_{\tilde{p}} \leq n_p$. Clearly,

$$\tilde{\phi}_i(\tilde{p}) \ \triangleq \ \phi_i(A_0\tilde{p} + b_0) \qquad 1 \leq i \leq m,$$

is also a convex function. Therefore the resulting optimization problem remains in the form of (**OP**) in (2.30).

Define the maximum design function $\Phi(\cdot)$ to be

$$\Phi(p) \ \triangleq \ \max_{1 \leq k \leq m} \phi_k(p). \qquad (3.1)$$

Clearly, if $\bar{p}$ is a feasible solution, we have

$$\Phi(\bar{p}) \leq 0.$$

Therefore, $\Phi(\cdot)$ can be viewed as the objective function in CODA.

The following two optimization approaches are used in CODA to solve the optimization problem (**OP**):

- descent-direction method

- barrier-function method

Strictly speaking, these two methods fall into the category of the so-called *method of centers* [7]. Both utilize the following conceptual algorithm:

A Conceptual Algorithm

Data: An initial design point $p_0 \in \mathbb{R}^{n_p}$.

Step 0: Set $k = 0$.

Step 1: If $\Phi(p_k) \leq 0$, stop.

Step 2: Find a search direction h_k.

Step 3: Solve the line search problem:

$$\bar{\lambda} = \arg \min_{\lambda > 0} \Phi(p_k + \lambda h_k).$$

Step 4: Let $p_{k+1} = p_k + \bar{\lambda} h_k$. Replace k by $k+1$ and go to Step 1.

Step 3 of solving the line search problem is implemented according to the Golden Section method [4, 10] in CODA. It is clear that Step 2 of finding a search direction plays the key role in the conceptual algorithm. The descent-direction and the barrier-function methods differ from each other mainly in the implementation of step 2. In the descent-direction algorithm, the search direction $\bar{h}$ is found by applying the nonsmooth analysis results [3, 11] so that

$$\bar{h} = \arg \max_{h} \frac{\Phi(p) - \Phi(p+h)}{\|h\|}. \tag{3.2}$$

In the barrier-function method, a standard convex logarithmic barrier function is used (see Section 3.3), and the *center* is defined to be the point minimizing the barrier function. The search direction is then chosen to be the vector from the current point to the center. It is clear from (3.2) that the descent-direction method deals directly with the objective function $\Phi(\cdot)$ in the search direction finding problem, while the search directions are found in the barrier-function method through the barrier function, which is not directly related to $\Phi(\cdot)$. On the other hand, the descent-direction method only utilizes the local (1st-order) information of $\Phi(\cdot)$, while the center in the barrier-function method is a global optimal solution of the associated barrier function, and usually provides more than 1st-order information about the objective function $\Phi(\cdot)$. Therefore, there exist trade-off performances between these two methods: the descent-direction method is more efficient in solving the problems with the MSV constraints while the barrier-function method is usually more efficient in solving the linear (time-domain constraints) and quadratic ($\mathcal{H}_2$ constraints) feasibility problems especially when the number of design requirements becomes bigger. The barrier-function method usually takes more CPU time in each iteration because another optimization cycle is invoked to find the center point, but may make a more significant move toward a feasible solution.

Because of the trade-off behaviors between the descent-direction and the barrier-function methods,

- a hybrid method

of these two approaches is implemented to share the advantages of both descent-direction and barrier-function methods.

The detail of these algorithms is introduced in the next sections. Again the notations in Section 2.5 are borrowed here.

3.2 Descent-Direction Method

To find a search direction for $\Phi(\cdot)$, define the ACDF (Augmented Convergent Direction Finding) map [11] by

$$G(p) \triangleq \mathrm{co}\,(\cup_{i=1}^{m} g_i(p)), \tag{3.3}$$

where co$(\cdot)$ denotes the convex hull of a set[4], and each $g_i(\cdot)$ is in the form of:

- for the RMS constraint — $g(\cdot) : \mathbb{R}^{n_p} \rightarrow \mathbb{R}^{n_p+1}$ with

$$g(p) = \begin{pmatrix} \Phi(p) - \phi(p) \\ \\ \frac{Hp+g}{2\phi(p)} \end{pmatrix}. \tag{3.4}$$

- for the time-domain constraint — $g(\cdot) : \mathbb{R}^{n_p} \rightarrow 2^{\mathbb{R}^{n_p+1}}$ with

$$g(p) = \mathrm{co}\begin{pmatrix} \Phi(p)\cdot \mathbf{1}^T - (\tilde{A}p + \tilde{b})^T \\ \\ \tilde{A}^T \end{pmatrix}, \tag{3.5}$$

  where $\mathbf{1}$ is an $(2n_s) \times 1$ vector with all of the elements equal to 1. Note that co(M) with M being a matrix indicates the convex hull of the column vectors of M.

[4]We say $g \in \mathrm{co}(G)$ with $G \subset \mathbb{R}^m$ if $\exists\, n > 0$, $\{d_k\}_{k=1}^{n} \subset G$ and $\{\mu_k\}_{k=1}^{n} \subset \mathbb{R}$ with $\mu_k > 0$, $\forall\, k$ and $\sum_{k=1}^{n} \mu_k = 1$ such that $g = \sum_{k=1}^{n} \mu_k d_k$. In short, co$(G)$ is the smallest convex set in $\mathbb{R}^m$ containing G.

- for the MSV constraint — $g(\cdot) : \mathbb{R}^{n_p} \to 2^{\mathbb{R}^{n_p+1}}$ with

$$g(p) = \text{co}\left\{ \left(\begin{array}{c} \Phi(p) - \psi(p,f,u,v) \\ \frac{\partial \psi}{\partial p}(p,f,u,v) \end{array} \right) \mid f \in \Omega, \ |u|_2 \le 1, \ |v|_2 \le 1 \right\}, \qquad (3.6)$$

where

$$\psi(p,f,u,v) \triangleq \text{Re}\,(u^*[\tilde{T}_{z_3w_3}(f) + \sum_{k=0}^{n_q} \tilde{d}_k(f)\tilde{T}_{z_3v}(f)Q_k(p)\tilde{T}_{ew_3}(f)]v) - \bar{b}(f). \qquad (3.7)$$

In (3.7), u and v are complex vectors with dimensions equal to those of w_3 and z_3 respectively, u^* denotes the complex conjugate transpose of u, and

$$\tilde{T}_{z_3w_3}(f) \triangleq \begin{cases} T_{z_3w_3}(j2\pi f), & \text{for continuous–time case;} \\ T_{z_3w_3}(e^{j2\pi T_0 f}), & \text{for discrete–time case.} \end{cases}$$

$\tilde{T}_{z_3v}(\cdot)$, $\tilde{T}_{ew_3}(\cdot)$ and $\tilde{d}_k(\cdot)$ are similarly defined from $T_{z_3v}(\cdot)$, $T_{ew_3}(\cdot)$ and $d_k(\cdot)$ respectively. With $\psi(\cdot)$ defined in (3.7), we have

$$\begin{aligned} \phi(p) &= \max_{f \in \Omega} \left(h(p,f) - \bar{b}(f) \right) \\ &= \max_{f \in \Omega} \left(\|\tilde{T}_{z_3w_3}(f) + \sum_{k=0}^{n_p} \tilde{d}_k(f)\tilde{T}_{z_3v}(f)Q_k(p)\tilde{T}_{ew_3}(f)\| - \bar{b}(f) \right) \\ &= \max_{\substack{f \in \Omega \\ |u|_2 \le 1 \\ |v|_2 \le 1}} \psi(p,f,u,v). \end{aligned}$$

Note that $\psi(\cdot)$ is differentiable in the design parameter p (but $\phi(\cdot)$ is not).

A search direction can be found by finding a solution to minimize the following semi-convex (quadratic) functions on the ACDF set $G(\cdot)$:

$$\bar{h}(p) \triangleq \left(\begin{array}{c} h_0(p) \\ -h(p) \end{array} \right) = \arg\min\{\xi_0 + \frac{1}{2}\xi^T Q \xi \mid \left(\begin{array}{c} \xi_0 \\ \xi \end{array} \right) \in G(p)\}, \qquad (3.8)$$

where ξ_0 and $h_0(\cdot)$ are scalars, ξ and $h(\cdot)$ are $n_p \times 1$ vectors, and Q is a positive-definite matrix currently chosen to be an identity matrix. The vector $h(\cdot)$ is then a descent direction for the objective function $\Phi(\cdot)$, which means that there exists a real number $\beta_0 > 0$ such that for any given positive number $\alpha < 1$,

$$\Phi(p + \beta h) - \Phi(p) \le -\alpha\beta|\bar{h}|_2^2, \qquad \forall\, 0 < \beta \le \beta_0.$$

The problem (3.8) can be solved in a very efficient way via the modified von Hohenbalkan algorithm [5, 6].

Some practical considerations on the implementation of the descent-direction algorithm can be found in [12].

3.3 Barrier-Function Method

Suppose the current design parameter is p_0. Let

$$\alpha \triangleq \Phi(p_0).$$

Define the barrier function $B(\cdot)$ to be

$$B(p) = \sum_{n=1}^{m} B_n(p), \tag{3.9}$$

where each $B_n(\cdot)$ is in the form of:

- for the RMS constraint:

$$\begin{aligned} B_n(p) &\triangleq -\ln(\alpha - \phi(p)) = -\ln(\alpha - F(p) + b_1) \\ &= -\ln(\alpha - \sqrt{\frac{1}{2}p^T H p + p^T g + f_0} + b_1). \end{aligned} \tag{3.10}$$

  It is straightforward to obtain

$$\frac{dB_n}{dp}(p) = \frac{\frac{1}{2}(Hp+g)}{(\alpha - \phi(p))F(p)}, \tag{3.11}$$

  and

$$\frac{d^2 B_n}{dp^2}(p) = \frac{\frac{1}{2}H}{(\alpha-\phi(p))F(p)} + \frac{\frac{1}{4}(Hp+g)(Hp+g)^T}{(\alpha-\phi(p))^2 F(p)^2} - \frac{\frac{1}{4}(Hp+g)(Hp+g)^T}{(\alpha-\phi(p))F(p)^3}. \tag{3.12}$$

- for the time-domain constraint: Consider the resulting linear inequality constraint in (2.20). Suppose

$$\tilde{A} \triangleq \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_{2n_s}^T \end{bmatrix}, \qquad \tilde{b} \triangleq \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{2n_s} \end{bmatrix},$$

  where each a_i is an $n_p \times 1$ vector, and b_i is a scalar. Define the associated barrier function to be

$$B_n(p) \triangleq -c_0 \sum_{i=1}^{2n_s} \ln(\alpha - a_i^T p - b_i), \tag{3.13}$$

  where c_0 is a weighting factor inversely proportional to the number of sampling points n_s. It follows from (3.13) that

$$\frac{dB_n}{dp}(p) = c_0 \sum_{i=1}^{2n_s} \frac{a_i}{\alpha - a_i^T p - b_i}, \tag{3.14}$$

  and

$$\frac{d^2 B_n}{dp^2}(p) = c_0 \sum_{i=1}^{2n_s} \frac{a_i a_i^T}{(\alpha - a_i^T p - b_i)^2}. \tag{3.15}$$

- for the MSV constraint: Without loss of generality, we assume that the MSV constraint is in the form of

$$\|T_0(f) + T_1(f)Q(p)T_2(f)\| - \bar{b}(f) \leq 0, \qquad \forall\, f \in \Omega, \qquad (3.16)$$

where $T_0(\cdot)$, $T_1(\cdot)$ and $T_2(\cdot)$ are $n_1 \times n_2$, $n_1 \times n_3$ and $n_4 \times n_2$ complex matrices respectively, and it implies that

$$Q(p) \in \mathbb{R}^{n_3 \times n_4}.$$

If T is an $n_1 \times n_2$ complex matrix, recall that

$$\|T\| = \max_{\substack{|u| \leq 1 \\ |v| \leq 1}} Re(u^* T v), \qquad (3.17)$$

where u and v are, respectively, $n_1 \times 1$ and $n_2 \times 1$ complex vectors, u^* denotes the complex conjugate transpose of the complex vector u, and $\mathrm{Re}(\cdot)$ is the real part of a complex argument. Let $R_k(\cdot)$ and $S_k(\cdot)$ denote the real and the imaginary parts of the complex matrix $T_k(\cdot)$, respectively, for $k = 0, 1, 2$ such that

$$T_k(f) \triangleq R_k(f) + j\, S_k(f), \qquad k = 0, 1, 2.$$

Define

$$\bar{R}(p, f) \triangleq R_0(f) + R_1(f)Q(p)R_2(f) - S_1(f)Q(p)S_2(f) \qquad (3.18)$$

and

$$\bar{S}(p, f) \triangleq S_0(f) + R_1(f)Q(p)S_2(f) + S_1(f)Q(p)R_2(f). \qquad (3.19)$$

We have

$$T_0(f) + T_1(f)Q(p)T_2(f) = \bar{R}(p, f) + j\, \bar{S}(p, f). \qquad (3.20)$$

Let

$$Z(p, f) \triangleq \begin{bmatrix} \bar{R} & -\bar{S} \\ \bar{S} & \bar{R} \end{bmatrix}. \qquad (3.21)$$

It is clear that $Z(p, f)$ is affinely linear in p, and

$$Z(p, f) \in \mathbb{R}^{(2n_1) \times (2n_2)}.$$

It follows from (3.17) and (3.20) that (3.16) is equivalent to

$$\|Z(p, f)\| - \bar{b}(f) \leq 0, \qquad \forall\, f \in \Omega, \qquad (3.22)$$

which can also be expressed in the standard form of the so-called linear matrix inequalities [1]

$$-\begin{bmatrix} \bar{b}(f) I_{2n_1} & Z(p, f) \\ Z^T(p, f) & \bar{b}(f) I_{2n_2} \end{bmatrix} \leq 0, \qquad \forall\, f \in \Omega. \qquad (3.23)$$

For each $f_i \in \Omega$, define

$$M_i(p) \triangleq \begin{bmatrix} (\alpha + \bar{b}(f_i))I_{2n_1} & Z(p, f_i) \\ Z^T(p, f_i) & (\alpha + \bar{b}(f_i))I_{2n_2} \end{bmatrix}. \tag{3.24}$$

Now we are ready to define the following barrier function for the MSV constraint:

$$B_n(p) \triangleq -c_0 \sum_{i=1}^{n_f} \ln[\det(M_i(p))], \tag{3.25}$$

where c_0 is the weighting factor which is inversely proportional to n_f, the number of frequency sampling points. It has been proved that the barrier function defined in (3.25) is convex [8].

To find the center for the barrier function, we need the first-order and the second-order derivatives of the barrier functions. For simplicity, we will study the case for a single item of $\{-c_0 \ln[\det(M_i(p))]\}$ in (3.25). Let

$$\bar{B} \triangleq -c_0 \ \ln[\det(M(p))].$$

Note that we have dropped the index of i for simplicity. Suppose p_k is the (k_1, k_2)-th element of the parameter matrix $Q(\cdot)$. Then

$$\frac{\partial \bar{B}}{\partial p_k}(p) = -c_0 \cdot tr(M(p)^{-1} \frac{\partial M}{\partial p_k}(p)). \tag{3.26}$$

It is clear that

$$\frac{\partial M}{\partial p_k}(p) = \begin{bmatrix} 0 & \frac{\partial Z}{\partial p_k} \\ \frac{\partial Z^T}{\partial p_k} & 0 \end{bmatrix}, \tag{3.27}$$

where

$$\frac{\partial Z}{\partial p_k} = \begin{bmatrix} R_1 e_{k_1} \bar{e}_{k_2}^T R_2 - S_1 e_{k_1} \bar{e}_{k_2}^T S_2 & -(S_1 e_{k_1} \bar{e}_{k_2}^T R_2 + R_1 e_{k_1} \bar{e}_{k_2}^T S_2) \\ S_1 e_{k_1} \bar{e}_{k_2}^T R_2 + R_1 e_{k_1} \bar{e}_{k_2}^T S_2 & R_1 e_{k_1} \bar{e}_{k_2}^T R_2 - S_1 e_{k_1} \bar{e}_{k_2}^T S_2 \end{bmatrix}, \tag{3.28}$$

with e_{k_1} ($\bar{e}_{k_2}$) denoting the k_1-th (k_2-th) column basis vector of the $n_3(n_4)$-dimensional Euclidean vector space, i.e., all of the elements equal to 0 except the k_1-th (k_2-th) one which is equal to 1. Note that $1 \leq k_1 \leq n_3$ and $1 \leq k_2 \leq n_4$.

To evaluate $M(p)^{-1}$, the following procedure is suggested: Let

$$M(p)^{-1} \triangleq \begin{bmatrix} M_1 & N \\ N^T & M_2 \end{bmatrix}.$$

If $n_1 \geq n_2$, let

$$T \triangleq ((\alpha + \bar{b}(f))^2 I_{2n_2} - Z^T Z)^{-1}, \tag{3.29}$$

then

$$\begin{aligned} M_2 &= (\alpha + \bar{b}(f)) \cdot T, \\ N^T &= -T \cdot Z^T, \\ M_1 &= (I_{2n_1} - Z \cdot N^T)/(\alpha + \bar{b}(f)); \end{aligned} \tag{3.30}$$

while if $n_1 < n_2$, let

$$T \triangleq ((\alpha + \bar{b}(f))^2 I_{2n_1} - ZZ^T)^{-1}, \tag{3.31}$$

then

$$\begin{aligned} M_1 &= (\alpha + \bar{b}(f)) \cdot T, \\ N^T &= -Z^T \cdot T, \\ M_1 &= (I_{2n_2} - N^T \cdot Z)/(\alpha + \bar{b}(f)). \end{aligned} \tag{3.32}$$

It follows from (3.26 - 3.32) that

$$\frac{\partial \bar{B}}{\partial p_k}(p) = -2c_0 \cdot tr(\frac{\partial Z}{\partial p_k} N^T). \tag{3.33}$$

Let

$$N^T \triangleq \begin{bmatrix} N_1 & N_2 \\ N_3 & N_4 \end{bmatrix},$$

where each $N_m \in \mathbb{R}^{n_1 \times n_2}$ for $1 \leq m \leq 4$. It is then straightforward to show that

$$\frac{\partial \bar{B}}{\partial p_k}(p) = -2c_0 \cdot (\bar{e}_{k_2}^T C e_{k_1}), \tag{3.34}$$

where

$$C = R_2(N_1+N_4)R_1 - S_2(N_1+N_4)S_1 + R_2(N_2-N_3)S_1 + S_2(N_2-N_3)R_1. \tag{3.35}$$

Therefore,

$$\frac{d\bar{B}}{dp}(p) = -2c_0 \cdot (C^T)(:), \tag{3.36}$$

where $C(:)$ denotes the vector of which the elements come from those of the matrix C column by column.

To evaluate the Hessian matrix for $\bar{B}(\cdot)$, we start at

$$\begin{aligned} \frac{\partial^2 \bar{B}}{\partial p_l \partial p_k}(p) &= c_0 \cdot tr(M(p)^{-1} \frac{\partial M}{\partial p_l}(p) M(p)^{-1} \frac{\partial M}{\partial p_k}(p)) \\ &= 2c_0 \cdot tr\left(\frac{\partial Z}{\partial p_l}(M_2 \frac{\partial Z^T}{\partial p_k} M_1 + N^T \frac{\partial Z}{\partial p_k} N^T)\right). \end{aligned} \tag{3.37}$$

Define

$$\begin{bmatrix} U_1 & U_2 \\ U_3 & U_4 \end{bmatrix} \triangleq \begin{bmatrix} R_1^T & I_1^T \\ -I_1^T & R_1^T \end{bmatrix} M_1 \begin{bmatrix} R_1 & -I_1 \\ I_1 & R_1 \end{bmatrix}, \tag{3.38}$$

$$\begin{bmatrix} V_1 & V_2 \\ V_3 & V_4 \end{bmatrix} \triangleq \begin{bmatrix} R_2 & -I_2 \\ I_2 & R_2 \end{bmatrix} M_2 \begin{bmatrix} R_2^T & I_2^T \\ -I_2^T & R_2^T \end{bmatrix}, \tag{3.39}$$

$$\begin{bmatrix} X_1 & X_2 \\ X_3 & X_4 \end{bmatrix} \triangleq \begin{bmatrix} R_2 & -I_2 \\ I_2 & R_2 \end{bmatrix} N^T \begin{bmatrix} R_1 & -I_1 \\ I_1 & R_1 \end{bmatrix}, \tag{3.40}$$

where the dimensions of $U_i's$, $1 \le i \le 4$, are the same, and so are the dimensions of $V_i's$ and the dimensions of $X_i's$. It is straightforward to show from (3.37) that

$$\frac{\partial\left(\frac{d\bar{B}}{dp}\right)}{\partial p_k}(p) = 2c_0 \cdot (C_k^T)(:), \qquad 1 \le k \le n_p, \tag{3.41}$$

where

$$\begin{aligned} C_k = {} & U_1 e_{k_1} \bar{e}_{k_2}^T V_1 + U_2 e_{k_1} \bar{e}_{k_2}^T V_3 + U_3 e_{k_1} \bar{e}_{k_2}^T V_2 + U_4 e_{k_1} \bar{e}_{k_2}^T V_4 + \\ & X_1 e_{k_1} \bar{e}_{k_2}^T X_1 + X_2 e_{k_1} \bar{e}_{k_2}^T X_3 + X_3 e_{k_1} \bar{e}_{k_2}^T X_2 + X_4 e_{k_1} \bar{e}_{k_2}^T X_4. \end{aligned} \tag{3.42}$$

Since the barrier-function defined in (3.9) is a convex function, there exists a global optimal point minimizing the barrier function, which is called the center of the barrier function. Since the first-order and the second-order information of the barrier functions are available, the Quasi-Newton method can be applied to find the center. The search direction in the step 2 of the conceptual algorithm in Section 3.1 is then equal to the vector from the current design point to the center.

3.4 A Hybrid Method

As mentioned in Section 3.1, there exist trade-off phenomena between the descent-direction and the barrier-function algorithms. The barrier-function algorithm takes more CPU time in each iteration because another optimization cycle is invoked to find the center, but may make a more significant move to a feasible solution. On the other hand, the search direction in the descent-direction algorithm is directly related to the objective function defined in (3.1), and is more efficient, especially, for the design problems with MSV constraints. To have the advantages of both algorithms together, a hybrid algorithm is implemented in CODA where the descent-direction and the barrier-function algorithms are applied alternately. Currently, an iteration using the barrier-function algorithm will be followed by five iterations using the descent-direction algorithm. It has been observed from practical design examples that the hybrid algorithm is very efficient especially for the design problems with MSV constraints and a significant number of step/impulse design requirements.

4 Interaction with ISIM and RealSim

4.1 Introduction

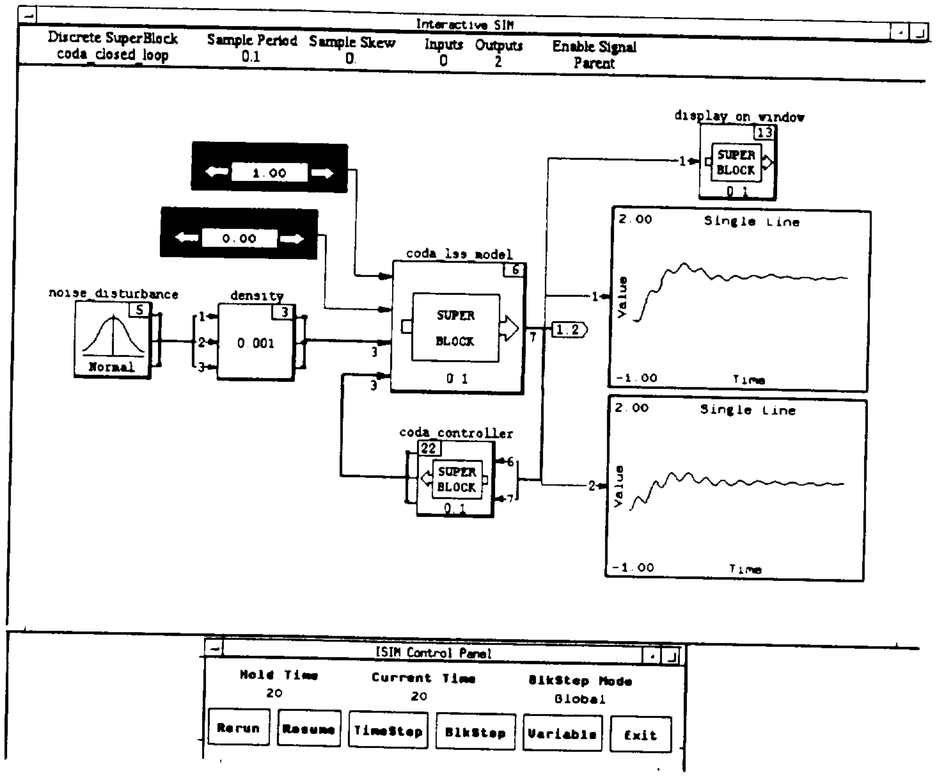
Simulation provides a direct and practical way to validate a controller design. In the MATRIX$_X$ CACE environment, two simulation facilities are available to test the controllers obtained from CODA:

- **ISIM** (Interactive SIMulation) module allows the user to test the controller via numerical simulation. The plant model given by the user in SystemBuild for simulation can be linear, nonlinear or even time-varying system which may be different from the linear time-invariant plant design model. To make the simulation model as close as possible to the practical one, appropriate noise, disturbance and/or plant uncertainty models should be included in the plant simulation model. As a linear dynamic model, the controller can be most conveniently realized by a state-space block in SystemBuild. However, it can also be realized by, e.g., a Block-Script block or a UCB (User-Code Block). The IA (Interactive Animation) icons can also be included to have interactively animated inputs and graphically displayed outputs. Various integration algorithms like Euler's method, Kutta-Merson methods, stiff-system solver, etc. are available for simulation. The default integration algorithm is the variable-step Kutta-Merson algorithm. The ISIM module can be invoked in Xmath by typing the following command in the Xmath command area:

  $$z \; = \; \text{sim}(\textit{model},\; t,\; w,\; \{\text{interact}\});$$

  where *model* is a string denoting the superblock name of the feedback system model in SystemBuild, t denotes the time variable which is a column vector, w indicates the external input variable and is a matrix of which the ith column corresponds the ith input, and z denotes the output variable which is a PDM (Parameter-Dependent Matrix) with domain equal to t. The argument w can be ignored if no external inputs exist. After the command is issued, an ISIM window along with the interactive control panel will appear, of which an example is shown in Figure 4.1. The control panel was originally designed to (i) specify a time for the simulation to pause by entering a number for the **hold time**; (ii) start, pause, resume, or exit the simulation process; etc.

- **RealSim** (Real-time Simulation) processor allows the user to simulate the controller on a real-time basis. The plant can be a hardware device such as an LSS, or a superblock model implemented in SystemBuild. The controller model in SystemBuild connected with the possibly existing plant model is turned into an executable C, ADA or Fortran code via the AutoCode. IA (Interactive Animation) icons can be included in the simulation model in the RealSim processor to interactively monitor the simulation results and control the magnitudes of the external input to the simulation model. HCE (Hardware Connection Editor) allows the user to connect the RealSim processor with any existing (plant)


Discrete SuperBlock: coda_closed_loop
Sample Period: 0.1
Sample Skew: 0
Inputs: 0
Outputs: 2
Enable Signal: Parent

Hold Time: 20
Current Time: 20
BlkStep Mode: Global

Figure 4.1: An ISIM example

hardwares. The graphical user interface of the RealSim processor is shown in Figure 4.2 from which it can be observed that the standard procedure of the RealSim process includes:

- Generate the real-time (*.rtf*) file for the simulation (controller) model in SystemBuild.
- Generate the C, ADA or Fortran executable code for the simulation model.
- Compile and link the executable code.
- Generate an IA diagram.
- Connect the AC-100 to associated hardwares.
- Download the executable code to the AC-100, and run the simulation.

CODA is aimed at providing not only a GUI multi-objective design environment, but also an automation tool for the interaction with the ISIM module and the RealSim processor. It is also desirable that the user can update the controller in the simulation model on an on-line basis. Two new features in MATRIX$_X$ 5.0 CACE environment help to reach the goals, which are SBA (SystemBuild Access) and RVE (Run-time Variable Editor). These two new features and their applications in CODA are discussed in the next two sections. A UCI (User-Callable Interface) allowing the user to display the ISIM results on the function windows of CODA is described in Section 4.4.

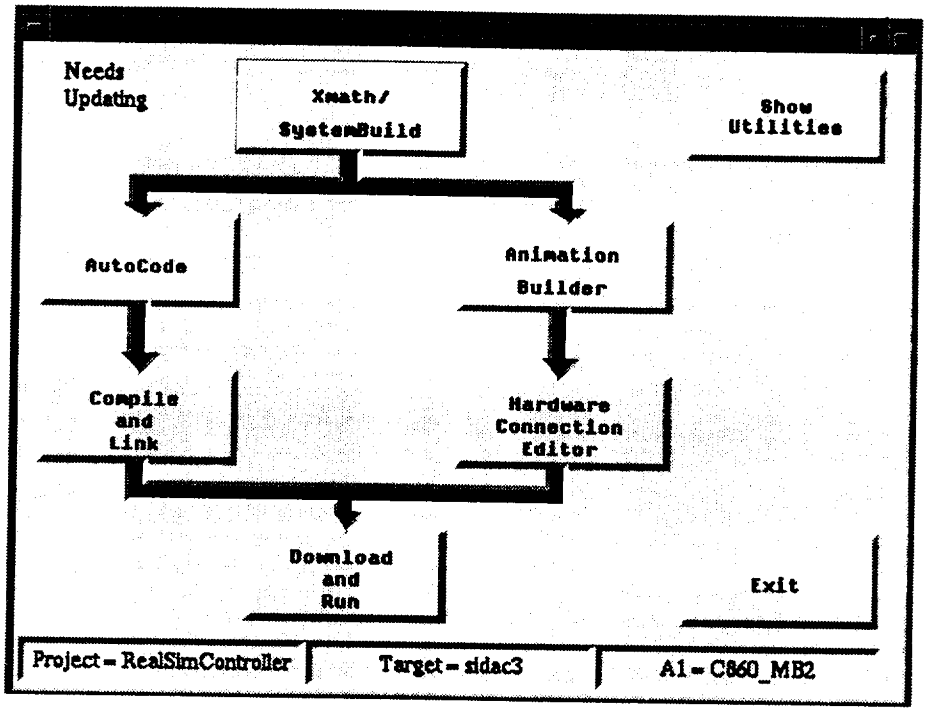


Figure 4.2: RealSim Graphical User Interface

4.2 Automated Modeling of Controller and Feedback Systems via SBA

The SBA feature provides an Xmath interface into the SystemBuild database. This interface is a collection of Xmath commands and functions that allow the user to create, modify, query, and delete SystemBuild diagrams. These commands and functions include, but are not restricted to:

$$\begin{array}{ll} \text{CreateSuperBlock} & \text{ModifySuperBlock} \\ \text{CreateBlock} & \text{ModifyBlock} \\ \text{DeleteSuperBlock} & \text{DeleteBlock} \\ \text{CreateConnection} & \cdots \end{array} \tag{4.1}$$

When the simulation phase is initiated in CODA:

1. A prototype controller model currently implemented in a BlockScript block will be loaded into the SystemBuild;
2. The SBA feature is then applied to
   - modify the parameter values in the controller model, such as the sizes of inputs, outputs and states, and the state-space matrices;
   - for the ISIM case, connect the controller to a plant simulation model given by the user to form a feedback model for simulation.

The controller or feedback model in SystemBuild is then ready for (i) ISIM, or (ii) starting the standard RealSim setup process shown in Figure 4.2.

Given a plant superblock model by the user for ISIM, the feedback simulation model in Figure 4.1 is constructed by CODA via the SBA commands in (4.1), except that the plant superblock has been expanded by the user via the 'Expand SuperBlock' button in the 'Edit' menu of SystemBuild.

4.3 Update of Controller Parameter via RVE

In a design cycle of CODA, various designs can be obtained and tested via the ISIM module or the RealSim processor. Previously, if a new controller was to be simulated, we had to

1. quit the current ISIM or RealSim session;
2. modify the controller model;
3. start a new simulation session;

which was very inefficient and would slow down the design cycle. As a new feature in MATRIX$_X$, RVE allows the user to change the %Variable values in many SystemBuild intrinsic blocks during the execution of an ISIM or RealSim test-bed session. Unfortunately, the state-space block and the UCB are currently not supported in RVE. The controller model is therefore implemented in the BlockScript block which is fully supported in RVE for both ISIM and RealSim cases.

The %Variables in the BlockScript block of the controller model indicate

- the state-space matrices of the nominal controller,
- the coefficient matrices in the Q expansion (see (2.5)),

which completely determine a controller model in CODA. Each time the user would like to update the controller design during simulation, he/she just has to push a button (see Figures 5.15 and 5.17) to invoke an updating functionality in CODA to update the %Variable values and, hence, the controller model in the ISIM or the RealSim process, via the Xmath RVE commands such as

rve_start, rve_put, rve_update, rve_stop.

The Xmath RVE commands

attach_ac100, rve_quit,

are also used for the RealSim case to connect or detach Xmath from the RealSim processor.

By utilizing RVE in CODA to interact with the ISIM module or the RealSim processor, the ISIM or the RealSim session has to be invoked only once for each design cycle, and the controller designs can be updated in the simulation process on an on-line basis. It helps to speed up the design cycle significantly.

Figure 1.1 provides a brief picture about updating the controller parameter via the RVE in CODA for both ISIM and RealSim processes.

4.4 ISIM Results Appearing on GUI Windows via a UCI

It is very desirable to have the simulation results displayed on the created time-domain function windows so that the user can monitor the simulation process by

- checking the simulation output values;
- comparing the simulation results with the time-domain responses or design specifications on the GUI windows.

To have the ISIM outputs displayed on the GUI windows:

1. A prototype UCB implemented in SystemBuild to serve as a UCI (User-Callable Interface) is loaded into SystemBuild;
2. The inputs of the UCB are then connected from the simulation outputs of interest via the SBA commands.

As an example, the UCB called 'display_on_window' in Figure 4.1 functions as a UCI for ISIM. What the UCB does during simulation includes:

- collecting the output data;
- writing the data to Xmath at a certain rate specified by the user.

The simulation outputs can then be displayed on the GUI windows as shown in Figure 4.3, which are characterized by gray curves. The user can then compare the simulation results with the time-domain responses or design specifications shown on the GUI windows, or check the output values by moving the cursor to the points of interest on the curve with the right button of mouse pushed.

The basic idea can be extended to the RealSim case except that the data will be acquired in Xmath via a TCP/IP-based socket (ACSOCK) connected to a RealSim processor. However, as of May 1996, this part remains to be finished.

Figure 1.1 provides a brief illustration about displaying the simulation results onto the GUI windows in CODA.

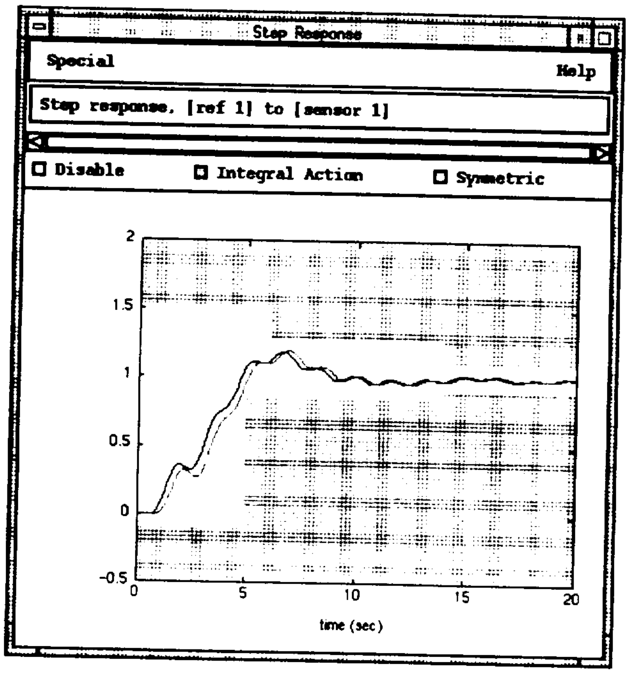


Figure 4.3: A GUI Window with Simulation Output Displayed

4.5 Conclusion

Utilizing the SBA and the RVE in Xmath, CODA provides a very convenient and powerful automation environment for the interaction with the ISIM module and the RealSim processor. The simulation model can be constructed automatically by CODA via the SBA feature, and the controller parameter can be updated on an on-line basis via the RVE facility. The UCB block served as a UCI for displaying the ISIM outputs onto the time-domain function windows makes CODA a even more handy tool for design.

5 Graphical User Interface

5.1 Introduction

As is clear from Figure 1.1, the user interacts with CODA only through the GUI facility in CODA. All of the other functionalities like preprocessing, optimization algorithms, SBA and RVE Mathscripts, etc. are transparent to the user, and are invoked by CODA itself as consequences of the user's interactions with CODA via the GUI. The GUI facility in CODA provides the following desirable features:

- In the design phase, the user's interaction with CODA basically stays at the level of designing performance specifications on a graphical basis. The GUI in CODA eliminates the need for the user to know the details of the underlying toolbox language, and design and optimization algorithms.
- In the simulation phase, the GUI facility is capable of interacting with the ISIM module or the RealSim processor, and implementing and modifying the simulation model on an automation and on-line basis. It helps to avoid invoking an individual ISIM or RealSim session for testing each new controller design.
- The GUI facility for CODA is designed to be intuitive, i.e., things mostly work the way the user would guess that they should work. The on-line help system provides all of the necessary and detailed information about using CODA as a multi-objective design tool. Furthermore, the intelligent message and warning system in CODA provides extensive status and error-detection messages to help the user to monitor and adjust the design procedure. It turns out that even the first-time user would find no difficulty using CODA for design.

Because of the user-friendly GUI facility, CODA appears to be a very effective, efficient and handy tool for design.

The GUI facility of CODA is built upon a fully programmable GUI design environment in Xmath. To use CODA, the user should:

- have a user's understanding of X-windows and the window manager that the user uses (e.g., the user should be able to move, resize, and iconify windows; use a pull-down menu; use a scroll bar),
- have a user's understanding of Xmath (enough to create a plant system object or a plant superblock model in SystemBuild, at least),
- know the basics of how to interact with an Xmath GUI application (e.g., use a slider to set a parameter value, a variable-edit box to type in a value, data-viewing and plot zooming),

- know the basics of control system design as a plus (e.g., the definitions of bandwidth, dB, stability, $\mathcal{H}_2$-norm (RMS) of a stable dynamic system, and the maximum singular value of a (complex) matrix).

An introduction to Xmath, and a basic introduction to X-windows, can be found in the Xmath Basics manual. There are several ways the user can find out about the basics of interacting with an Xmath GUI application:

- A short discussion appears in several places, e.g., Chapter 1 of the Xmath GUI Reference Manual, and Chapter 8 of the Xmath Basics Manual (v4.0).
- Typing "help GUI" in the Xmath command window gives a good introduction.
- Typing "guidemo" in the Xmath command window allows the user to start up and play with several GUI demo applications; this allows the user to try out sliders, pushbuttons, scroll bars, dataviewing, and so on, as the user reads about them.

Once the user has mastered the basic mechanics of using an Xmath GUI application, the user should be ready to get started. To start up CODA, type

```
coda
```

in the Xmath command window. After the CODA main window appears, the Xmath command prompt will return. The user can now use Xmath and CODA simultaneously. The user can get a good overview of the features of CODA by scanning the entries in the menu bars and reading the help messages in the help menu of the main window.

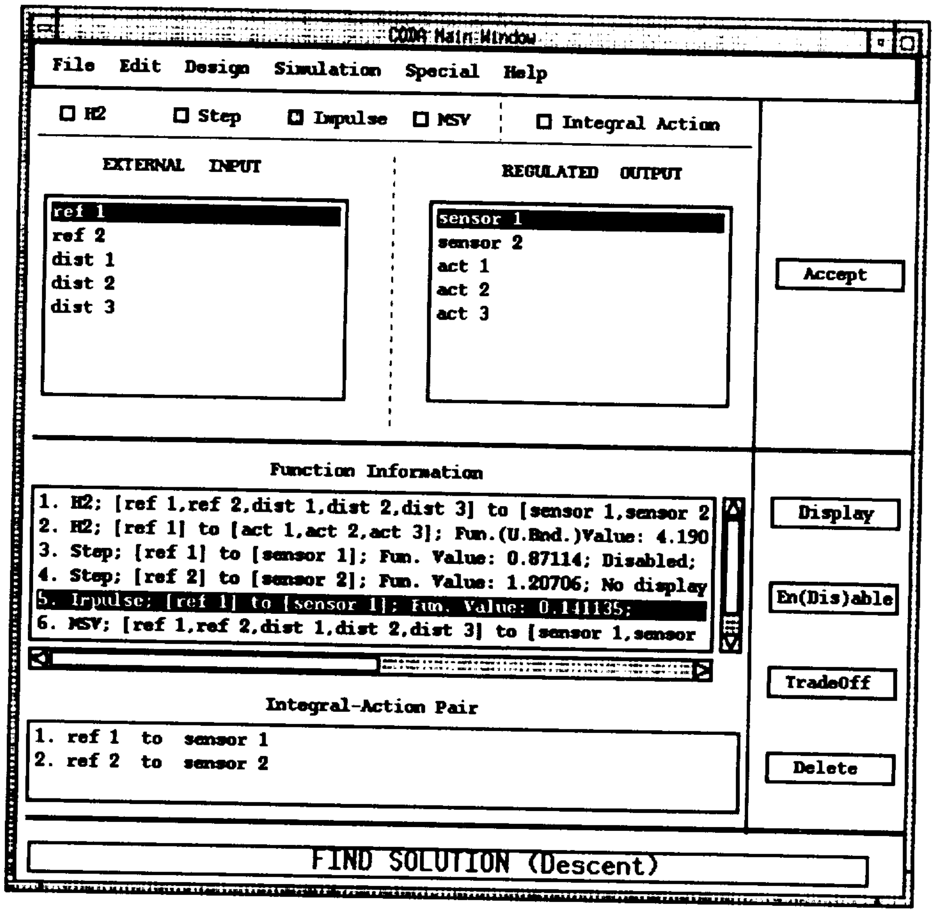
It is worthwhile to mention that the basic CODA architecture shown in Figure 1.1 and the flow chart of a typical design cycle shown in Figure 1.2 provide an overall idea about how CODA functions and how it can be fully utilized in control system design. It is recommended that the first-time user go over these two diagrams first before using CODA.

The remainder of this section gives a general descriptions on the GUI facility of CODA.

5.2 CODA Main Window

A CODA main window is shown in Figure 5.1. It consists of, from top to bottom:

- A menu bar with **File**, **Edit**, **Design**, **Simulation**, **Special**, and **Help** menus;



ref 1
ref 2
dist 1
dist 2
dist 3

sensor 1
sensor 2
act 1
act 2
act 3

Function Information

```
1. H2; [ref 1,ref 2,dist 1,dist 2,dist 3] to [sensor 1,sensor 2
2. H2; [ref 1] to [act 1,act 2,act 3]; Fun.(U.Bnd.)Value: 4.190
3. Step; [ref 1] to [sensor 1]; Fun. Value: 0.87114; Disabled;
4. Step; [ref 2] to [sensor 2]; Fun. Value: 1.20706; No display
5. Impulse; [ref 1] to [sensor 1]; Fun. Value: 0.141135;
6. MSV; [ref 1,ref 2,dist 1,dist 2,dist 3] to [sensor 1,sensor
```

Integral-Action Pair

```
1. ref 1  to  sensor 1
2. ref 2  to  sensor 2
```

Figure 5.1: A CODA main window

- A collection of togglebuttons for selecting the function type ($\mathcal{H}_2$, Step, Impulse, MSV or Integral Action), two extended-select scrolled lists for selecting the function inputs and outputs, and the pushbutton **Accept** to preprocess the function; Note that,

    1. after the button **Accept** is pushed, a window will appear for each new well-defined function with given default bound(s), and the function values (for $\mathcal{H}_2$ functions) or the responses (for step, impulse, or MSV functions) displayed for the most recent 3 designs (see Section 5.3). Note that the function window can be closed by the user.
    2. to define multiple integral pairs at one shot, select the inputs first and then the corresponding outputs in pair on the list order of the selected inputs.
    3. if the types of step or impulse is chosen, multiple inputs and multiple outputs are allowed, which will result in a multiple-function window (see Section 5.3.2).
    4. the step function and the integral-action pair can be defined at the same time. Integral-action pairs can also be added or removed from step windows (see Section 5.3).

- Two extended-select scrolled lists for functions and integral-action pairs, which contain the informations about the functions and the integral-action pairs; The information for each function on the function list includes:

    1. the function type;
    2. the function input and output;
    3. the RMS value and the upper bound for the $\mathcal{H}_2$ functions;
       the design function value (defined in (2.21) and (2.25)) for the step, impulse and MSV functions;
    4. the message of 'No display' if the function values or responses are currently not displayed on any GUI window;
    5. the message of 'Disabled' if the function is currently disabled, which means it is not counted as one of design specifications.

- Four pushbuttons which are

    1. **Display**: to create displaying windows for the functions selected from the function list of which the corresponding windows have been closed (see Section 5.3).
    2. **En(Dis)able**: to enable (disable) the selected functions as parts of the design specifications if they are currently disabled (enabled). Note that the design specifications are characterized by the design functions which are not disabled. A new created function is not disabled by default. The functions can be disabled or enabled via the GUI function windows as well (see Section 5.3).

3. **TradeOff**: to create a trade-off window for any two functions selected from the scrolled lists (see Section 5.3.3).
4. **Delete**: to delete the functions or integral-action pairs selected from the scrolled lists.

- A pushbutton '**Find Solution** (with the name of the chosen optimization algorithm)' for finding a solution to satisfy the design specifications; To switch to another optimization algorithm, just choose another optimization algorithm via the **Optimization** cascade button in the **Design** menu (see Section 5.2.3).

Other than the **help** menu, the menus on the menu bar of CODA main window are introduced in the rest of this section.

5.2.1 File Menu

The **File** menu is used to communicate with Xmath, i.e., read plants and design parameters from Xmath or SystemBuild, and write design parameters, controllers or nominal controllers from CODA back to Xmath.

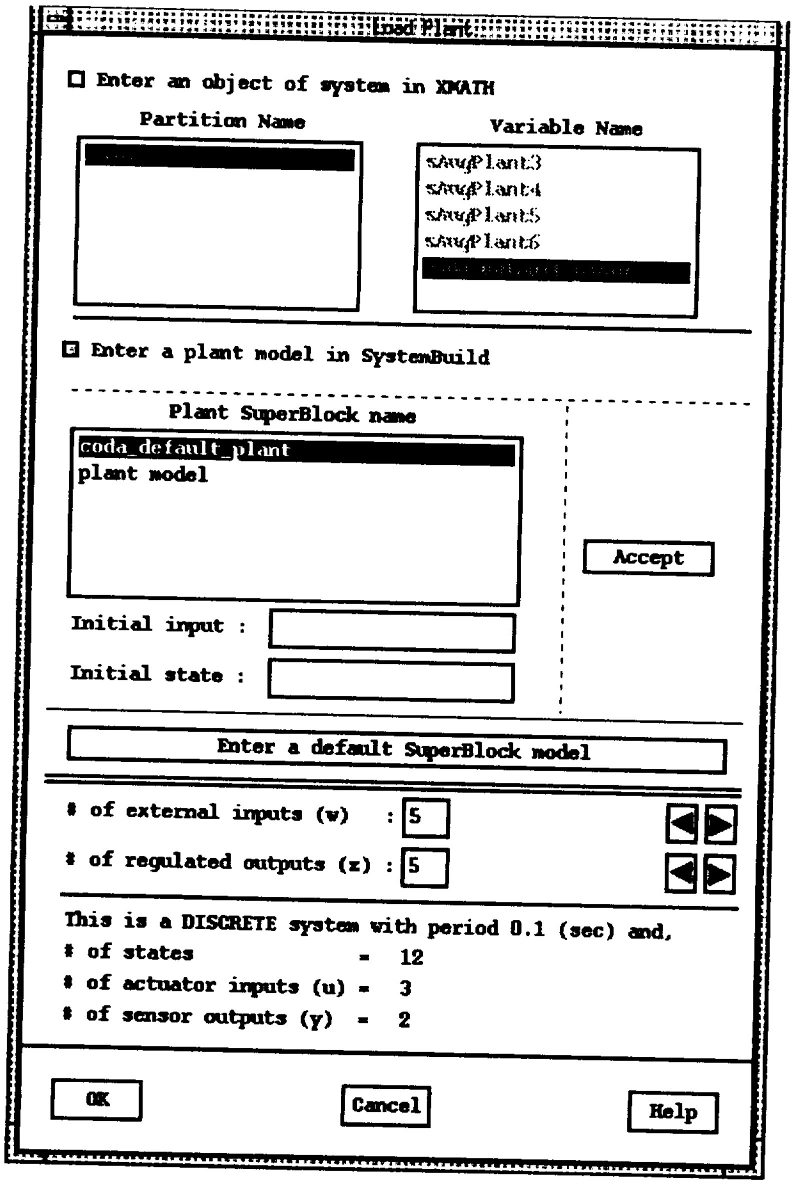
Load Plant

The first thing the user has to do after CODA is invoked is to load a 4-block plant (see Figure 2.1) into CODA. A window will pop up for this purpose (see Figure 5.2), which provides two ways for the user to input a plant design model:

1. Select an object of system in Xmath. This can be done by first selecting the partition name where the object resides, and then the object name from the single-select scrolled lists. Note that all of the variables listed are objects of system.
2. Select a superblock model in SystemBuild, which needs not to be linear. The user can also input the variable names for the initial input and state with which a linearized model can be obtained from the selected model for design after the **Accept** button is pushed. By default, the initial input and state vectors are zeros.

For each of the two cases, CODA provides a default plant model which is meant to be used when the user is learning to use CODA and needs a quick way to enter a plant. Each plant model is derived from an LSS model. The default superblock model is a linear discrete model which has the model diagram shown in Figure 5.3 with

$$\begin{array}{llll} \#\text{ of states} = 12, & & & \\ \#\text{ of external inputs}\,(w) = 5, & & \#\text{ of regulated outputs}\,(z) = 5, & \\ \#\text{ of actuator inputs} = 3, & & \#\text{ of sensor outputs} = 2. & \end{array}$$


Load Plant

☐ Enter an object of system in XMATH

Partition Name	Variable Name
[illegible]	sAugPlant3
	sAugPlant4
	sAugPlant5
	sAugPlant6
	[illegible]

☐ Enter a plant model in SystemBuild

Plant SuperBlock name

coda_default_plant
plant model

Accept

Initial input :

Initial state :

Enter a default SuperBlock model

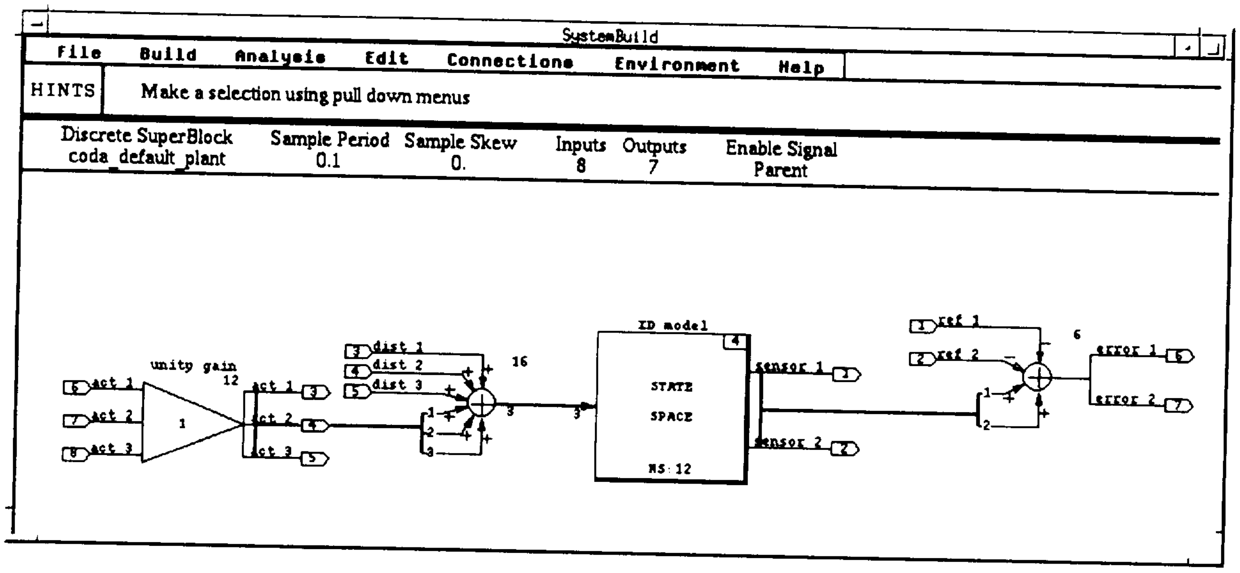
# of external inputs (w)   : 5

# of regulated outputs (z) : 5

```
This is a DISCRETE system with period 0.1 (sec) and,
# of states                =  12
# of actuator inputs (u)   =  3
# of sensor outputs (y)    =  2
```

OK   Cancel   Help

Figure 5.2: The widget for loading a plant design model



HINTS Make a selection using pull down menus

Discrete SuperBlock coda_default_plant | Sample Period 0.1 | Sample Skew 0. | Inputs 8 | Outputs 7 | Enable Signal Parent

Figure 5.3: The default plant superblock model in SystemBuild

The external input consists of 2 reference inputs and 3 disturbance inputs while the regulated output is composed of 2 sensor outputs and 3 actuator inputs. The default system object indicates a continuous system with

$$\begin{array}{ll} \#\text{ of states } = 20, & \\ \#\text{ of external inputs}\,(w) = 5, & \#\text{ of regulated outputs}\,(z) = 5, \\ \#\text{ of actuator inputs } = 2, & \#\text{ of sensor outputs } = 3, \end{array}$$

and has a model diagram similar to that shown in Figure 5.3.

After the plant model is entered, the user has to specify the dimensions of the external input (w) and the regulated output (z) to completely determine the plant model. As can be seen from Figure 5.2, the information about the augmented 4-block plant model like the type of the plant and the sizes of states, actuator inputs and sensor outputs is provided for the user.

Read/Write Q design parameter from/to Xmath

The user can read the current design parameter from Xmath or write it from CODA to Xmath. Referred to (2.5), the design parameter is stored in the form of

$$[Q_0 \quad Q_1 \quad \dots \quad Q_{n_q}]$$

if the controller is not strictly proper, or

$$[Q_1 \quad Q_2 \quad \dots \quad Q_{n_q}]$$

if the controller is strictly proper.

Write Nominal Controller to Xmath

The nominal controller can be written back to Xmath for the other design purposes. It can be represented by either of the following two ways (see Figure 2.2 and (2.1)):

1. the state-feedback gain and the output-injection gain;
2. a 4-block system object.

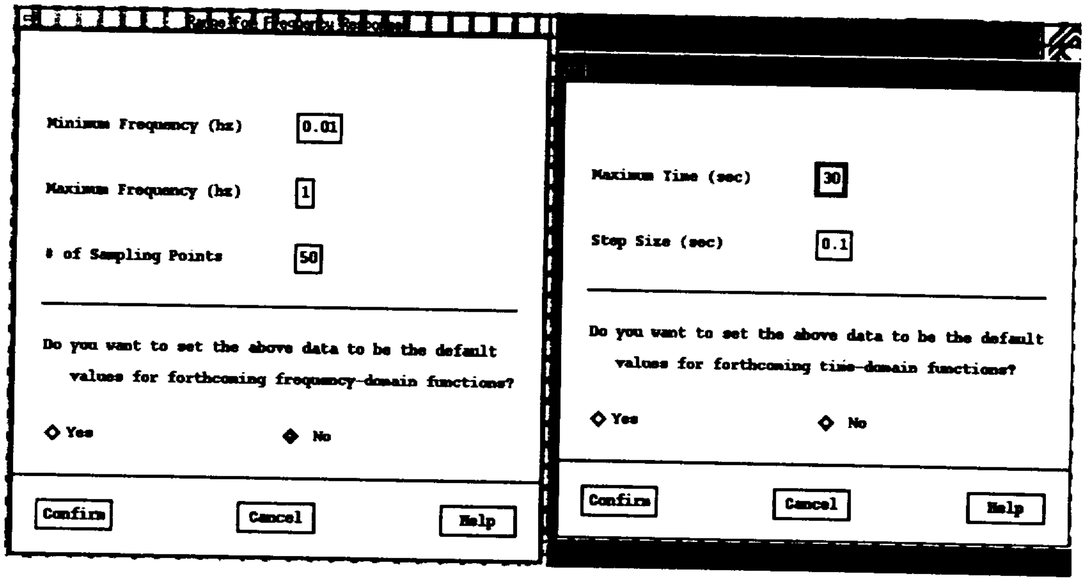
Note that if a set of state-feedback and output-injection gains stabilizes two different plant models simultaneously, it will result in two different nominal controllers for these two plants according to (2.1).

5.2.2 Edit Menu

The Edit menu is used to (i) change the function ranges, (ii) assign a new Q expansion, and (iii) store the design parameter to the design pool or select the designs from the pool.

Change Function Ranges

In CODA, the user can change (see Figure 5.4)



Minimum Frequency (hz) 0.01

Maximum Frequency (hz) 1

# of Sampling Points 50

Do you want to set the above data to be the default values for forthcoming frequency-domain functions?

◇ Yes [✓ No]

Maximum Time (sec) 30

Step Size (sec) 0.1

Do you want to set the above data to be the default values for forthcoming time-domain functions?

◇ Yes ◇ No

Figure 5.4: The widget to change function ranges for MSV functions and step/impulse functions

- the maximum time span. Default = 20 (sec).
- the sampling rate (for the continuous-time case only). Default = 0.1 (sec).

for step and impulse functions; and

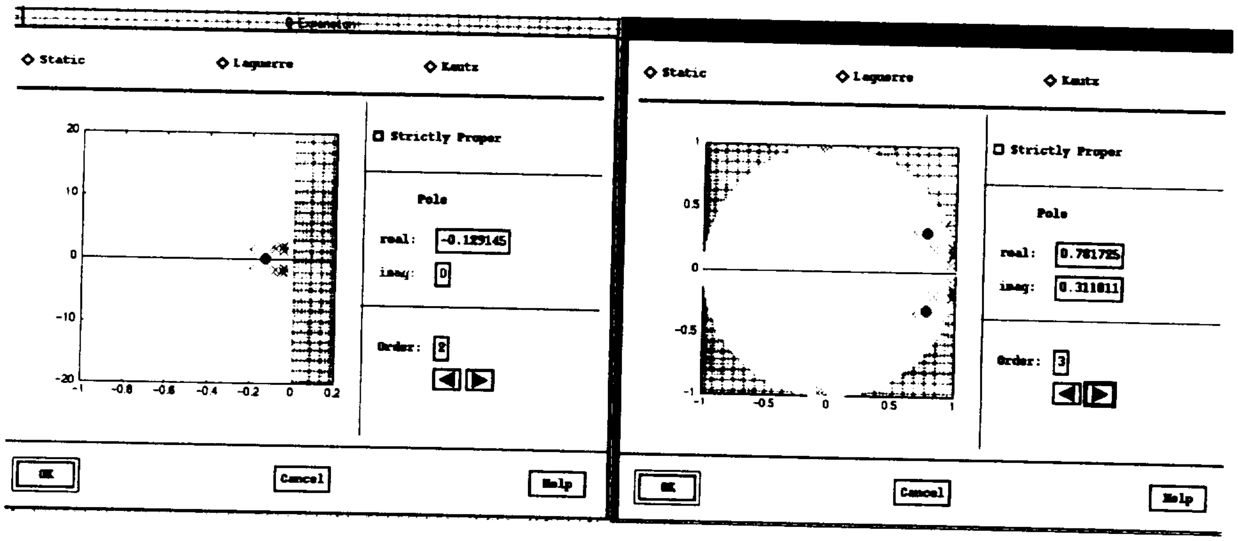
- the minimum and maximum frequencies. Defaults are $[0.01, 10]$ (Hz) for the continuous-time case, and $[\frac{0.001}{2T_0}, \frac{1}{2T_0})$ (Hz) for the discrete-time case where T_0 denotes the sampling rate.

- the # of sampling frequency points. Default $= 30$.

for MSV functions. The user can change function ranges either for the functions of the same types selected from the function list or for the function to be created with the function type has been chosen. The user can also have the new range setup become the default one for the functions to be defined.

New Q-Expansion

The user can assign new Q expansions for design. The widget used to determine the Q expansion and hence the parameter space is shown in Figure 5.5 where:



□ Strictly Proper

Pole

real: -0.129145

imag: 0

Order: 2

□ Strictly Proper

Pole

real: 0.781725

imag: 0.311811

Order: 3

Figure 5.5: Q-expansion widgets: Left – for the continuous-time case; Right – for the discrete-time case

- there are 3 types of expansion (see Section 2.3):

  1. Static: The Q dynamics is reduced to a static transfer matrix, i.e., (2.5) is reduced to

     $$Q(p, \cdot) \equiv Q_0(p).$$

     This is the default Q expansion.

  2. Laguerre expansion

  3. Kautz expansion

- the pole of the Q expansion can be assigned by

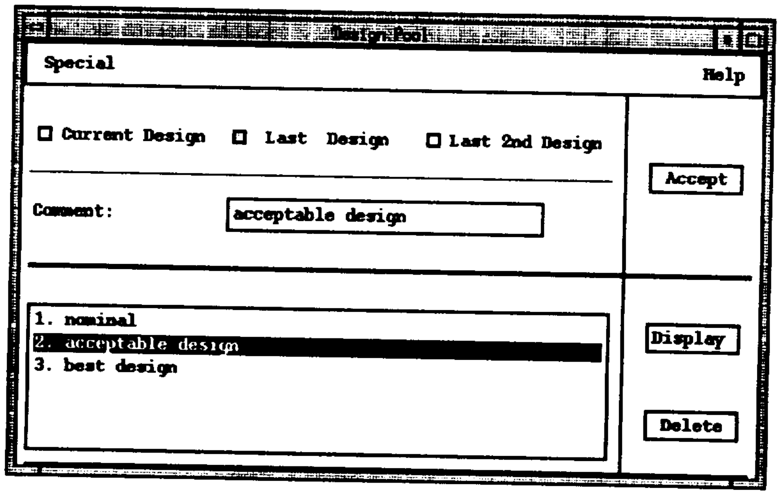
1. either entering numbers for the real and imaginary parts of the pole. Note that the imaginary part is fixed to be equal to zero for the case of Laguerre expansion.
2. or moving the cursor on the complex plane to the desired position with the <control> key and the left button of the mouse pushed.

Since the Q dynamics should be stable, its pole should be assigned on the open left half plane for the continuous-time case, or within the unit circle for the discrete-time case. The pole of the Q expansion is marked by the solid dark circle(s) in the complex plane. Note that the poles and the zeros of the nominal closed-loop system marked by 'x' and 'o', respectively, in the light color are shown for references.

- the order of the expansion can be assigned by either entering a number for it or pushing the 'increase' or 'decrease' buttons.

Design Pool

In the widget of design pool shown in Figure 5.6, the user can



Comment: acceptable design

1. nominal
2. acceptable design
3. best design

Figure 5.6: The design pool

- store the design parameter to the design pool with a tag on it by (i) selecting one of the most recent 3 design parameters, (ii) giving a comment like nominal design, best design, etc. to it, and (iii) pushing the Accept button;
- select the design from the list of designs, and
  1. push the Display button. The function values or responses corresponding to the selected design will be displayed on the GUI function windows to be introduced in Section 5.3.

2. push the **Delete** button to remove the design from the design pool. Note that more than one designs can be selected and then deleted at one time.
3. change the tag by typing a new comment and then pushing the **Accept** button.

It is important to note that the design pool will be cleared up whenever

1. a new plant is loaded to CODA;
2. a new nominal controller is obtained;
3. the pole of the Q-expansion is changed;
4. the dimension of design parameter is decreased by either reducing the order of Q-expansion, or adding new integral-action pairs.

5.2.3 Design Menu

The **Design** menu provides an interactive design environment where the user can:

- find a nominal controller;
- obtain a minimum-mean-square solution for the time-domain functions or the $\mathcal{H}_2$ functions;
- do the trade-off analysis between two designs by taking the convex combinations of the corresponding two parameter points, and obtain a 'best compromised' design from it;
- change the optimization algorithms;
- invoke a performance-meter window to monitor the values of design functions during the design process.

CODA keeps track of the function values or responses for the most recent 3 design points during the design process. These 3 designs called the current design, the last design and the last 2nd design are denoted by p_0, p_1 and p_2 respectively.

Nominal Controller Design

After the plant is loaded, the user has to find a stabilizing nominal controller before going ahead to define the design specifications and do the Q-parameter design. The user can change the nominal controller during the design process. CODA provides 6 options for the user to find a (4-block) nominal controller:

1. Read a 4-block nominal controller from Xmath as an object of system with $(n_u + n_y)$ inputs and $(n_u + n_y)$ outputs (see Figure 2.2 and Equation (2.1)).

2. Set the state-feedback and output-injection gains to be zeros if the plant is stable. The preprocessing scheme will be simplified for this special case.

3. Read the stabilizing state-feedback and output-injection gains from Xmath as objects of matrices with appropriate dimensions. If the default discrete-time plant model is used, CODA provides a set of stabilizing gains, 'coda_regu_gain' and 'coda_esti_gain', in Xmath for the user to obtain a stabilizing 4-block nominal controller from (2.1).

4. Apply the LQG method to the given plant model, i.e., find a 4-block nominal controller by finding optimal state-feedback and output-injection gains to minimize the $\mathcal{H}_2$-norm (RMS) from w to z for the plant model.

5. Apply the LQG method to a plant model obtained from the plant design model with selected external inputs and regulated outputs. This is for the case that some external inputs and/or regulated outputs, like the reference input, may not be relevant and should not be included in determining the regulator and estimator gains.

6. Apply the LQG method to a simplified model shown in Figure 5.7, where P_{yu} denotes the one-block sub-model of the plant from the actuator input (u) to

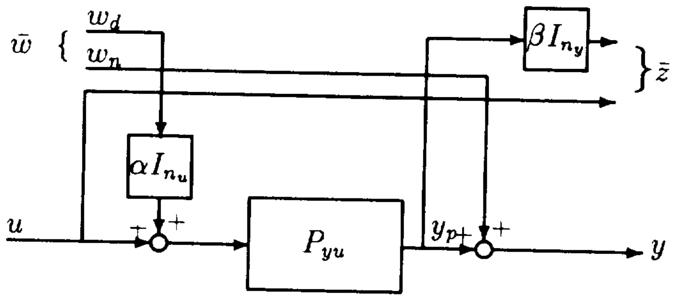
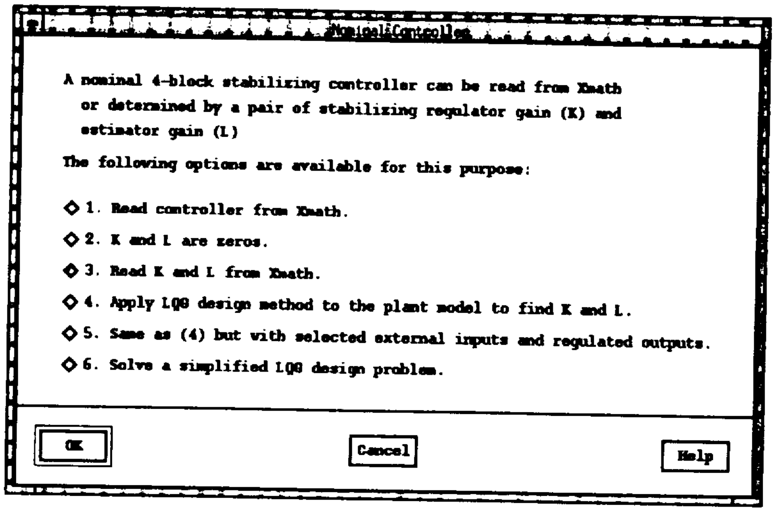


Figure 5.7: The simplified model for LQG design

the sensor output (y), the external input $\bar{w}$ consists of the actuator disturbance w_d and the output noise w_n, the regulated output $\bar{z}$ comes from the actuator input u and the plant output y_p, α indicates the ratio of the actuator disturbance intensity and the sensor noise intensity, and β denotes the ratio of the weighting factors on the plant output and the actuator input. Given an actuator-to-sensor plant model P_{yu}, any positive scalars α and β uniquely determine the output-injection and the state-feedback gains respectively. Note that the model in Figure 5.7 is derived from the 4-block plant design model given by the user only for obtaining a nominal controller, and will be removed after that.

The window widget with the above 6 options is shown in Figure 5.8. If the user choose the option of (1), (3), (5) or (6), a separate window will appear to ask for the relevant information to determine a 4-block nominal controller.



A nominal 4-block stabilizing controller can be read from Xmath or determined by a pair of stabilizing regulator gain (K) and estimator gain (L)

The following options are available for this purpose:

◇ 1. Read controller from Xmath.

◇ 2. K and L are zeros.

◆ 3. Read K and L from Xmath.

◇ 4. Apply LQG design method to the plant model to find K and L.

◇ 5. Same as (4) but with selected external inputs and regulated outputs.

◇ 6. Solve a simplified LQG design problem.

Figure 5.8: The widget for nominal controller design

Each time a new nominal controller is obtained during the design process, the preprocessing scheme will be invoked to reprocess the functions have been created based on the current function ranges, Q-expansion, and integral-action pairs. The current design parameter p_0 will be reset to be a zero vector, and the parameters p_1 and p_2 will be set to be null vectors.

Minimum-Mean-Square Problems

The multi-objective design problem is basically a min-max optimization problem (see the **OP** problem in (2.30), and (3.1)) which doesn't have an analytic solution. A feasible solution can only be found by an optimization algorithm on an iterative basis. However, some minimum-mean-square problems may be derived from the design problem, which have analytic solutions. The analytic solutions may serve as good initial points for the optimization algorithms to find a solution. The following two types of minimum-mean-square problems are considered in CODA:

1. Suppose there are m $\mathcal{H}_2$ functions $\{F_i(\cdot)\}_{i=1}^m$ expressed in the form of (see (2.12))

$$F_i(p) = \sqrt{\frac{1}{2}p^T H_i p + p^T g_i + f_i}, \qquad 1 \le i \le m.$$

   Then there exists an analytic solution to the sum of the squares of the functions:

$$\min_p \sum_{i=1}^{m} F_i^2(p).$$

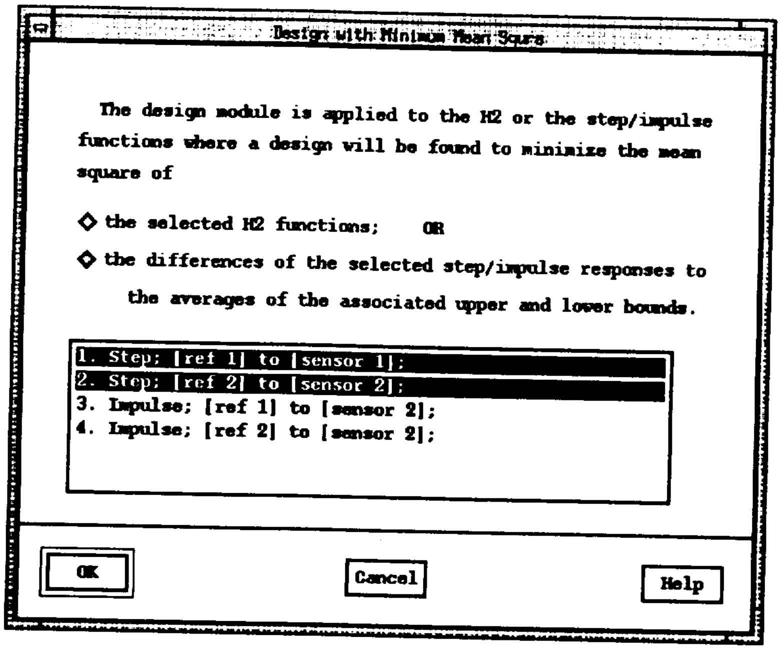
2. Consider the m step/impulse responses expressed by (see (2.17))

$$A_i p + b_i, \qquad 1 \le i \le m$$

with upper bounds $\{\bar{b}_i\}_{i=1}^m$ and lower bounds $\{\underline{b}_i\}_{i=1}^m$. There exists an analytic solution to the following minimum-mean-square problem:

$$\min_p \sum_{i=1}^m \mid A_i p + b_i - \frac{\bar{b}_i + \underline{b}_i}{2} \mid_2^2 .$$

The widget used for this purpose is shown in Figure 5.9, where the user has the


Design with Minimum Mean Square

The design module is applied to the H2 or the step/impulse functions where a design will be found to minimize the mean square of

◇ the selected H2 functions; OR

◇ the differences of the selected step/impulse responses to the averages of the associated upper and lower bounds.

1. Step; [ref 1] to [sensor 1];
2. Step; [ref 2] to [sensor 2];
3. Impulse; [ref 1] to [sensor 2];
4. Impulse; [ref 2] to [sensor 2];

Figure 5.9: The widget for defining minimum-mean-square problems

freedom of choosing the functions to define the minimum-mean-square problem.

Convex Combinations of Two Design Points

When this functionality is invoked, the last design will become the last 2nd design, and the current design will become the last design, i.e.,

$$p_1 \rightarrow p_2, \qquad p_0 \rightarrow p_1,$$

and the current design point p_0 will be set to be

$$p_0 = (1 - \lambda)p_1 + \lambda p_2, \qquad 0 \leq \lambda \leq 1.$$

The user can change the value of λ by moving the slider's handle or entering a number as shown in Figure 5.10. Each time the user moves the slider's handle, the current design point p_0 will change, and the corresponding function values and/or responses will be updated on the GUI function and trade-off windows (see Section 5.3). An example is shown in Figure 5.11 where the solid point or curve displayed in the black color indicates the function values or the step response corresponding to the design

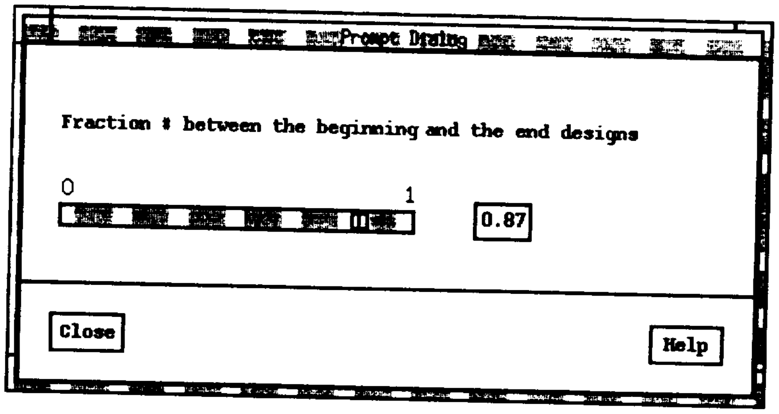


Figure 5.10: The widget for convex combinations of two design points

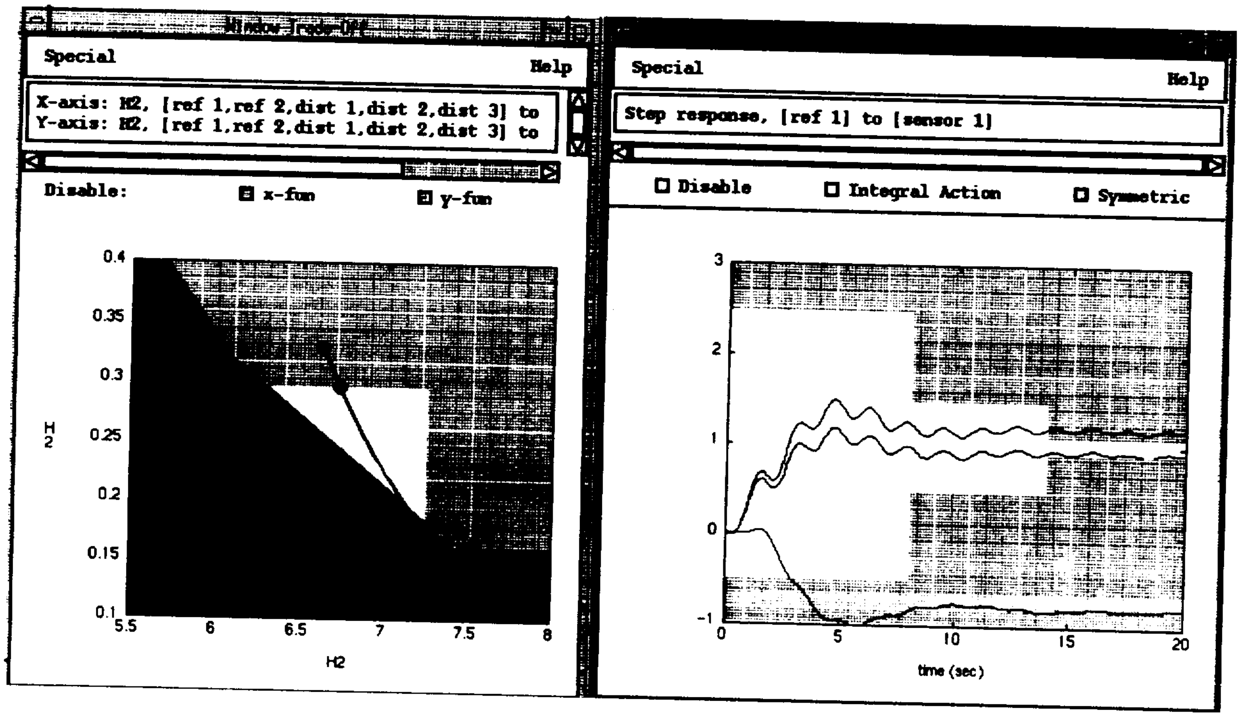


Figure 5.11: Two interactive windows: right - An $\mathcal{H}_2$ trade-off window; left - A step window

point in (5.2.3) with $\lambda = 0.87$, and will move if the value of λ changes. Note that the RMS trajectory with λ varying between 0 and 1 is shown on the $\mathcal{H}_2$ trade-off window.

Optimization Algorithms

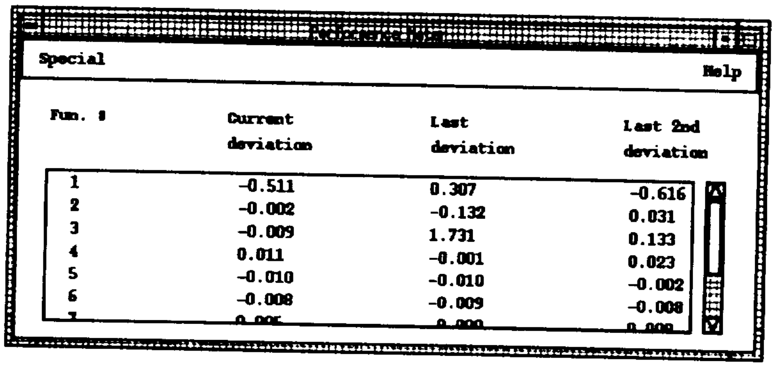
The user can select one of the following 3 optimization algorithms as the solver for design problems from the cascade button **Optimization Algorithm**:

1. Descent-Direction Method
2. Barrier-Function Method
3. Hybrid Method

The name of the selected optimization algorithm will appear in the pushbutton **Find Solution** on the bottom of the CODA main window (see Figure 5.1).

Performance Meter

The performance meter displays the design function values defined in (2.14), (2.21) and (2.25) for the most recent 3 designs. It allows the user to observe how close the design points are to the feasible region, and hence is helpful to monitor the design process. An example of the performance meter is shown in Figure 5.12.



Fun. #	Current deviation	Last deviation	Last 2nd deviation
1	-0.511	0.307	-0.616
2	-0.002	-0.132	0.031
3	-0.009	1.731	0.133
4	0.011	-0.001	0.023
5	-0.010	-0.010	-0.002
6	-0.008	-0.009	-0.008
7	[illegible]	[illegible]	[illegible]

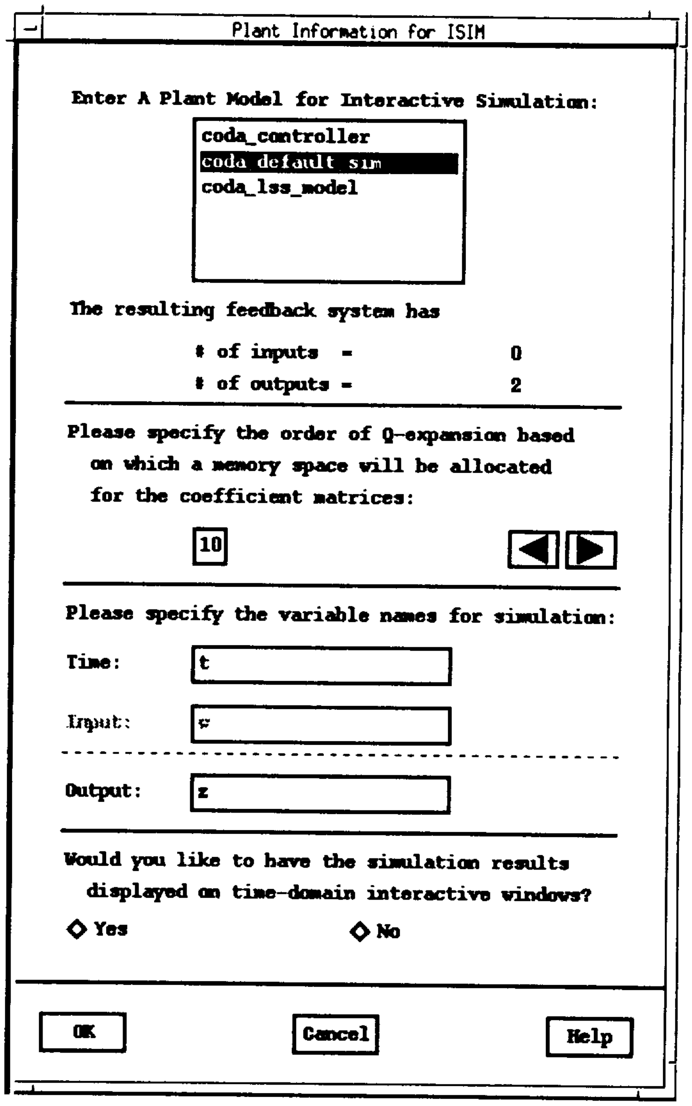
Figure 5.12: A performance meter

5.2.4 Simulation Menu

The Simulation menu allows the user to interact with the ISIM module and the Real-Sim processor to test the controller designs on an on-line basis.

Interactive Simulation

As a first step for the interaction with the ISIM module, a window will appear as shown in Figure 5.13 to ask the user to:



Plant Information for ISIM

Enter A Plant Model for Interactive Simulation:

coda_controller
coda default sim
coda_lss_model

The resulting feedback system has

# of inputs =	0
# of outputs =	2

Please specify the order of Q-expansion based on which a memory space will be allocated for the coefficient matrices:

10

Please specify the variable names for simulation:

Time:	t
Input:	v
Output:	z

Would you like to have the simulation results displayed on time-domain interactive windows?

◇ Yes ◇ No

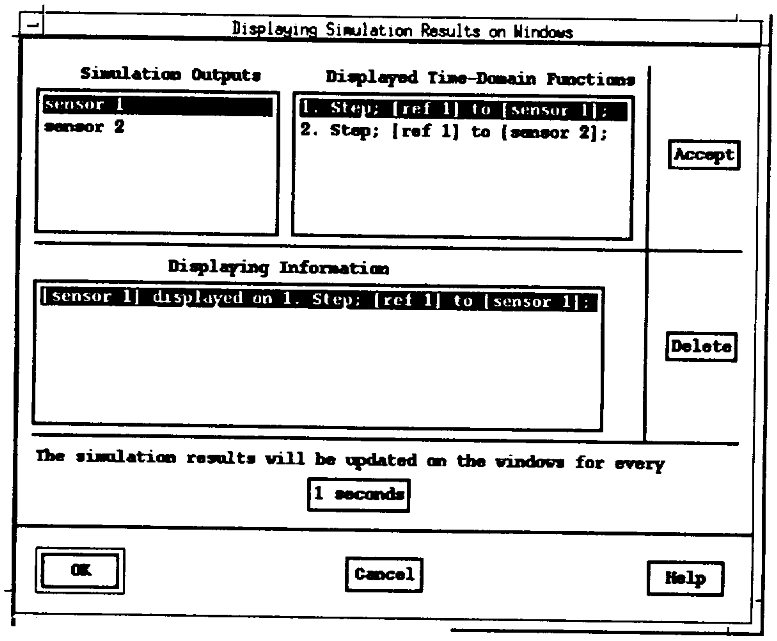
Figure 5.13: A GUI widget to acquire informations for ISIM

- select a plant simulation model in SystemBuild. If a default plant is used for design, a default plant simulation model will be provided by CODA for simulation. The default plant simulation model is the same as the default design model except that some IA icons may be attached to the external inputs and the regulated outputs of the plant simulation model.

- enter the order of the Q expansion based on which a memory space is allocated for the controller model so that as long as the order of Q expansion used in design is no larger than the one entered here, the user can modify the controller models in the ISIM module on an on-line basis.

- specify the time variable and the external input variable (if applicable) in Xmath as input arguments for ISIM, as well as the output variable in Xmath to which the simulation outputs will write.

- decide if the simulation results will be displayed on the step/impulse windows in CODA. If the answer is positive, a separate window as shown in Figure 5.14 will pop up for the the user to decide:


Displaying Simulation Results on Windows

Simulation Outputs
sensor 1
sensor 2

Displayed Time-Domain Functions
1. Step; [ref 1] to [sensor 1];
2. Step; [ref 1] to [sensor 2];

Accept

Displaying Information
[sensor 1] displayed on 1. Step; [ref 1] to [sensor 1];

Delete

The simulation results will be updated on the windows for every
1 seconds

OK
Cancel
Help

Figure 5.14: A GUI widget for acquiring informations on displaying simulation outputs on step/impulse windows

1. the outputs to be displayed.
2. the time-domain windows on which the outputs will be displayed. Note that more than one outputs can be displayed on a window.
3. the rate at which the simulation outputs will be updated on the step or impulse windows.

After the required information is provided, the feedback simulation model will be automatically constructed by CODA utilizing the SBA feature. The user then can modify the feedback model, e.g., expand the plant superblock via the 'Expand SuperBlock' button in the 'Edit' menu of SystemBuild, add or delete IA icons, etc. After this, an ISIM windows with a control panel will appear (see Figure 4.1). A GUI widget as shown in Figure 5.15 will show up to allow the user to update the controllers in the simulation model during simulation.

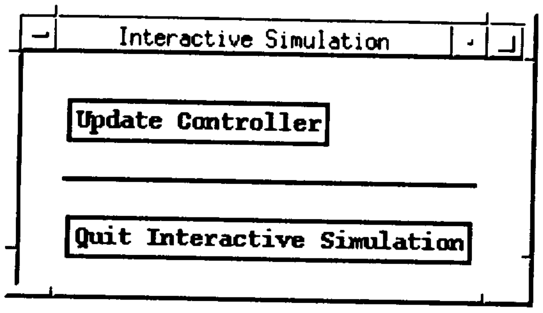


Figure 5.15: A GUI widget for updating the controllers in the interactive simulation process

Real-Time Simulation

The procedure for setting up the interaction channel with a RealSim processor is similar to the ISIM case except that:

- The simulation model initially consists of the controller model only. The user is not asked to supply with a plant model because the controller model may be connected to a hardware plant model. However, similar to the ISIM case, the user can modify the simulation model in SystemBuild before a real-time file (.rtf) is created for it, and it is the time that the user can connect the controller model to a plant model in SystemBuild to form a (feedback) simulation model for real-time simulation.

- The simulation outputs currently cannot be displayed on the time-function windows.

- After the .rtf file is created for the simulation model, the user has to go over the standard RealSim process shown in Figure 4.2 once. After that, CODA will link the AC-100 with the Xmath session. A widget created by the RealSim processor will appear as shown in Figure 5.16 to indicate that the communication channel has been established between the Xmath session and the AC-100. Note that the RealSim/AC-100 project directory should be the same as the current Xmath working directory. Similar to the ISIM case, a GUI widget will appear as shown in Figure 5.17 to allow the user to update the controllers in the RealSim processor during simulation.

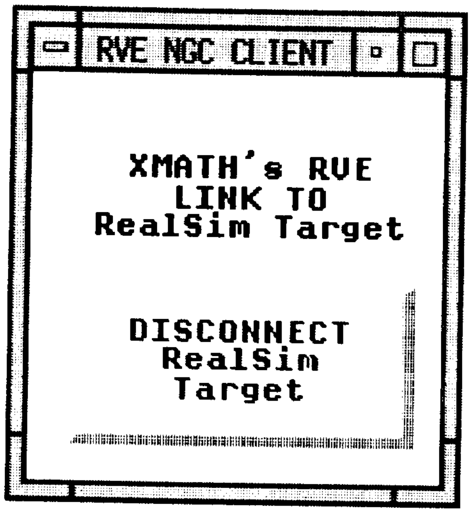


Figure 5.16: The widget indicating the linkage between the Xmath session and an AC-100

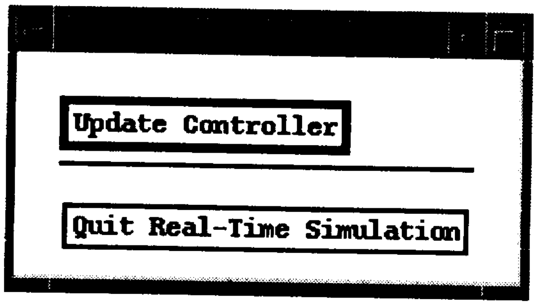
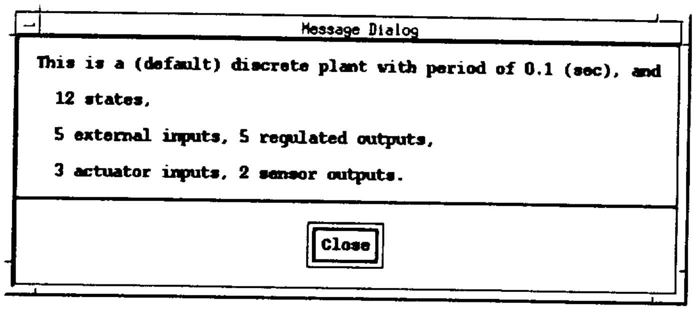


Figure 5.17: A GUI widget for updating the controllers in the RealSim processor

5.2.5 Special Menu

The special menu is currently used to provide:

- the information on the plant design model as a reminder to the user about the plant for which CODA is applied (see Figure 5.18).



Message Dialog

This is a (default) discrete plant with period of 0.1 (sec), and
12 states,
5 external inputs, 5 regulated outputs,
3 actuator inputs, 2 sensor outputs.

Figure 5.18: A widget showing the plant information

- the pole-zero plot for the closed-loop system corresponding to the current design. An example is shown in Figure 5.19 where the poles and the zeros are marked

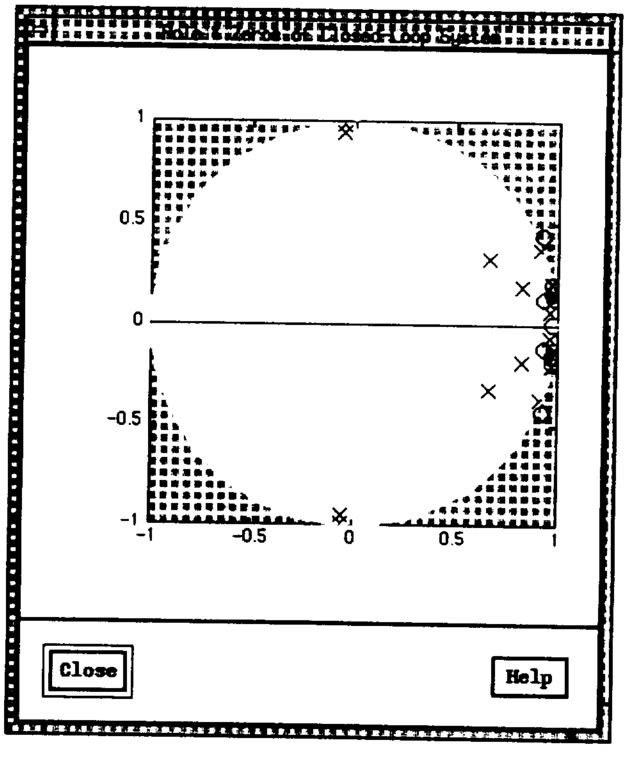


Figure 5.19: A widget showing the pole-zero plot of the closed-loop system corresponding to the current design

by 'x' and 'o' respectively. The locations of the poles and zeros can be read by pointing the cursor to the point of interest with the right button of the mouse pushed.

5.3 Interactive Function and Trade-off Windows

After the user selects the function type (H2, Step, Impulse or MSV), input and output, and pushes the **Accept** button on the CODA main window (see Figure 5.1), an interactive window with default bound(s) will show up. After the bounds are modified, the corresponding design function (see (2.14), (2.21) and (2.25)) is well-defined, and becomes a part of the design specifications unless the function is disabled. The windows also allow the user to monitor the function values and responses during the design process.

If a window is created for each function, too many windows may appear on the screen. CODA allows more than one functions of the same type share a single window. This helps to keep the number of windows created to the minimum.

It is worthwhile to mention that each function can be displayed on at most one window which is either a single-function window or a multiple-function window.

A trade-off window can be created for any two functions selected from the function list on the CODA main window. It allows the user to study and monitor the trade-off behavior between two functions, and is very useful in determining the appropriate performance specifications for design. For the case that both two functions are $\mathcal{H}_2$ functions, a trade-off curve will appear on the window, which provides very useful information on the trade-off behavior between these two $\mathcal{H}_2$ functions.

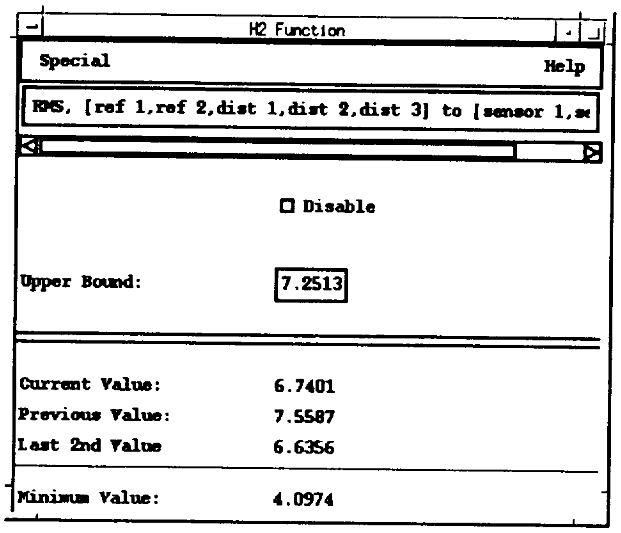
The interactive windows will be discussed in more detail in the rest of the section.

5.3.1 Interactive Function Windows

$\mathcal{H}_2$ Windows

An $\mathcal{H}_2$ window is shown in Figure 5.20 where:

- the user can input the upper bound (which is b_1 in (2.14)). The default upper bound is the five times of the minimum value.
- the function values corresponding to the most recent 3 designs, as well as the minimum RMS value are shown.
- the function information is shown in the scrolled text area below the menu bar of the window.
- the status of activeness of the function as a part of design specifications is controlled by the **Disable** togglebutton.



RMS, [ref 1,ref 2,dist 1,dist 2,dist 3] to [sensor 1,se

Upper Bound:	7.2513
Current Value:	6.7401
Previous Value:	7.5587
Last 2nd Value	6.6356
Minimum Value:	4.0974

Figure 5.20: An $\mathcal{H}_2$ interactive window

- in the Special menu, the user can
  1. obtain the design achieving the minimum RMS value.
  2. close the window. After the window is closed, the function remains active as one of the design specifications as long as it is not disabled.

Step/Impulse Windows

A step and an impulse windows are shown in Figure 5.21, where:

- in the Special menu,
  1. the user can copy the bounds on the window for the other functions of the same type.
  2. for the step response, the user can change the asymptotic value which is equal to 1 by default. Both upper and lower bounds will be shifted vertically by the difference between the new and the old asymptotic values. For the impulse response, the asymptotic value is fixed to be zero.
  3. close the window. After the window is closed, the function remains active as one of the design specifications as long as it is not disabled.
- the function information is shown in the scrolled text area below the menu bar of the window.
- the status of activeness of the function as a part of design specifications is controlled by the Disable togglebutton; the togglebutton Symmetric determines if the upper and the lower bounds are symmetric to the asymptotic value. For the step response, the user can push the Integral Action button to achieve the

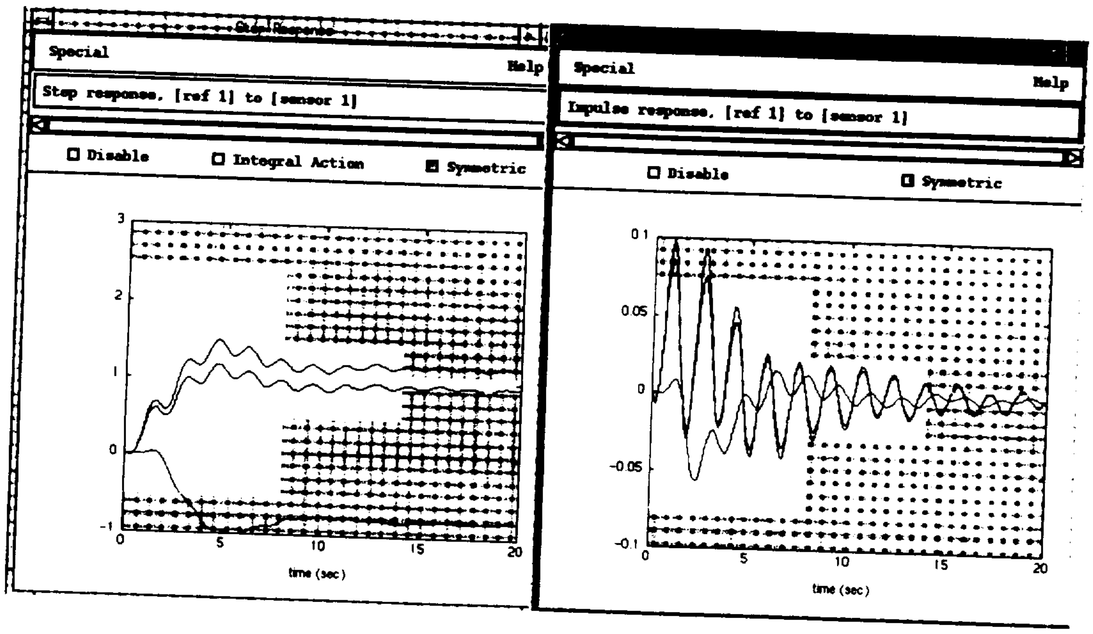


Figure 5.21: A step and an impulse windows

tracking requirement for the corresponding input-output pair, i.e., the steady-state value of the step response will be equal to the asymptotic value specified by the user.

- the undesired region is displayed in the color of lavender. The user can modify the bounds by pointing the cursor to any of the line segments of the bounds and dragging it with the <Control> key and the left button of the mouse pushed. A widget as shown in Figure 5.22 will appear for the user to confirm or cancel the changes of the bounds.

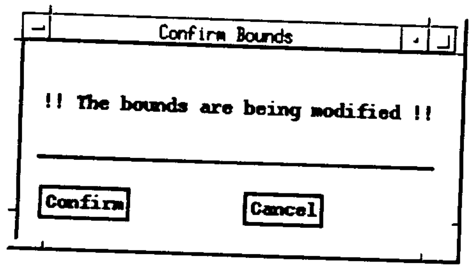


Figure 5.22: A widget to confirm or cancel the changes of the bounds

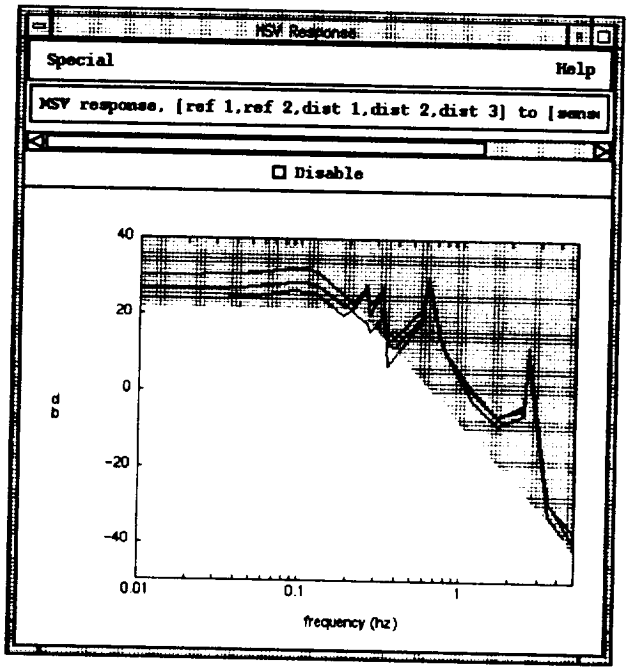
- the responses associated with the most recent 3 designs will be shown, which are plotted in the colors of black, red and blue for the current, the last and the last 2nd design respectively.
- the user can re-scale the plot by moving the cursor to any of the boundaries or corners of the plot area, and dragging it with the left button of the mouse pushed. However, the x (time) axis cannot be re-scaled to be over the maximum

time of the step or impulse response. The user is referred to the Function Range button in the Edit menu on the main window for the change of the time span.

- the values of points on the curves can be displayed by pointing the cursor to the point of interest on the curves with the right button of the mouse pushed.

MSV Window

A MSV window is shown in Figure 5.23 where:



MSV response, [ref 1,ref 2,dist 1,dist 2,dist 3] to [sens

Figure 5.23: A MSV window

- in the Special menu, the user can
  1. copy the bound on the window for the other MSV functions.
  2. close the window. After the window is closed, the function remains active as one of the design specifications as long as it is not disabled.
- the function information is shown in the scrolled text area below the menu bar of the window.
- the status of activeness of the function as a part of design specifications is controlled by the Disable togglebutton;
- the undesired region is displayed in the color of lavender. The user can modify the bounds by pointing the cursor to one of the line segments or the cut-off point on the bound, and dragging it with the <Control> key and the left button of

the mouse pushed. It allows the user to specify the desired gain, bandwidth and rolling rate for the MSV response. Similar to the step/impulse function, a widget as shown in Figure 5.22 will appear for the user to confirm or cancel the changes of the bound.

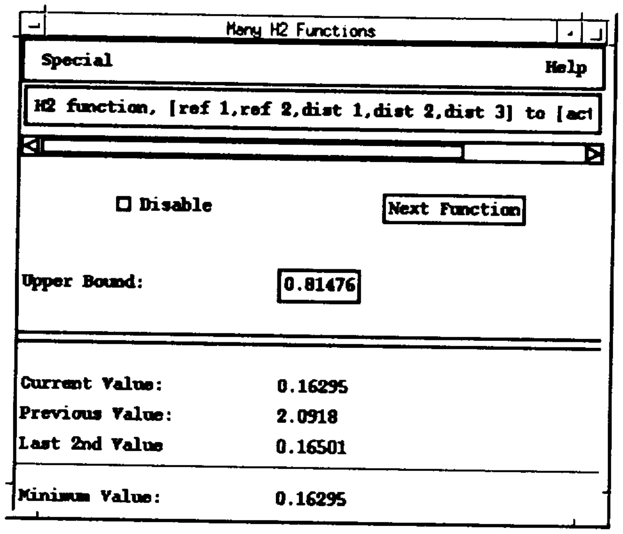
- similar to the step/impulse case, the user can re-scale the plot by moving the cursor to one of the boundaries or corners of the plot area, and dragging it with the left button of the mouse pushed. However, the x (frequency) axis cannot be re-scaled to be over the current frequency span of the MSV response. The user is referred to the **Function Range** button in the **Edit** menu on the main window for the change of the frequency span.
- the values of the curves can be displayed by pointing the cursor to the point of interest on the curves with the right button of the mouse pushed.

5.3.2 Multiple-Function Windows

To avoid too many windows appearing on the screen, CODA allows more than one functions of the same type share a single window.

Multiple-$\mathcal{H}_2$ Windows

To create a multiple-$\mathcal{H}_2$ window, the user (i) first selects the undisplayed $\mathcal{H}_2$ functions from the function list on the main window, and (ii) then pushs the **Display** button. A multiple-$\mathcal{H}_2$ window is shown in Figure 5.24 which looks and functions similar to the



H2 function, [ref 1,ref 2,dist 1,dist 2,dist 3] to [act

Upper Bound: 0.81476

Current Value: 0.16295
Previous Value: 2.0918
Last 2nd Value 0.16501

Minimum Value: 0.16295

Figure 5.24: A multiple-$\mathcal{H}_2$ interactive window

$\mathcal{H}_2$ window in Figure 5.20 except that:

- only the information of an $\mathcal{H}_2$ function is displayed on the multiple-$\mathcal{H}_2$ window. To display another function, the user can

1. either push the **Next Function** button,
2. or select the function to be displayed from the function list on the main window and push the **Display** button there.

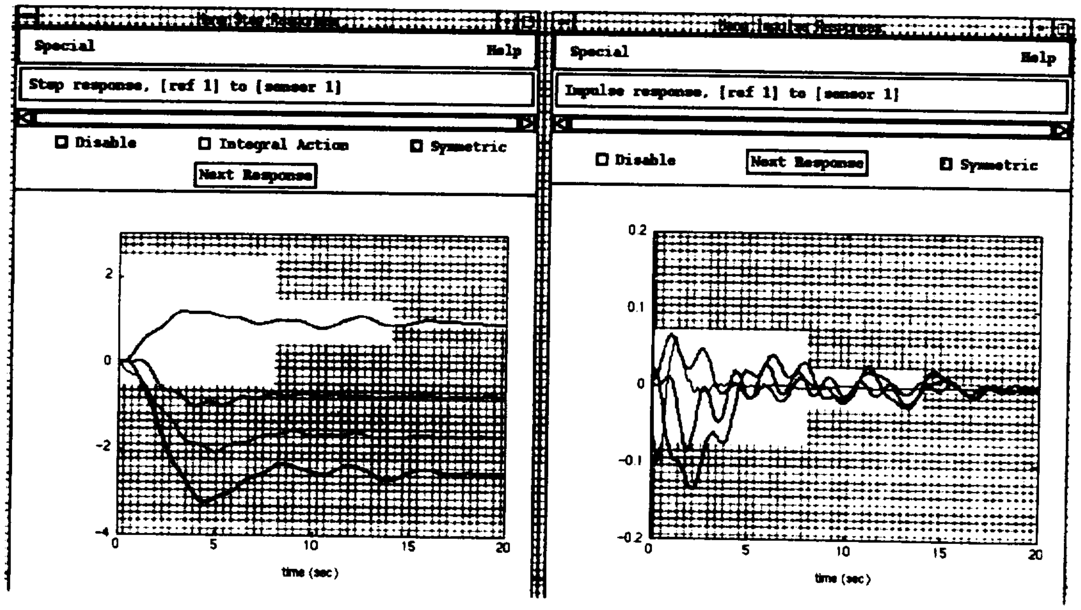
- in the **Special** menu, the user can remove functions from the window, or add other $\mathcal{H}_2$ functions to the window.

Multiple-Step and Multiple-Impulse Windows

There are two ways that the user can create a multiple-step/impulse window:

1. In the process of defining new functions, the user can choose multiple inputs and/or multiple outputs in the main window with the **Step** or **Impulse** toggle-button on, and then push the **Accept** button. The responses corresponding to all possible combinations of the inputs and the outputs will be displayed on a single window.
2. Similar to the $\mathcal{H}_2$ case, the user (i) first chooses the undisplayed step (impulse) functions from the function list on the main window, and (ii) then pushes the **Display** button.

A multiple-step and a multiple-impulse windows are shown in Figure 5.25 which looks and functions similar to Figure 5.21 except that:



Step response, [ref 1] to [sensor 1]

Impulse response, [ref 1] to [sensor 1]

Figure 5.25: A multiple-step and a multiple-impulse windows

- all of the responses of the functions corresponding the current design are displayed. However, only one of the responses is highlighted. The responses of the highlighted function corresponding to the last and the last 2nd designs are displayed and highlighted too. Only the bounds of the highlighted function are shown with the undesired region displayed in the color of lavender. Similar to the single-function case, the responses corresponding to the current, the last and the last 2nd designs are plotted in the colors of black, red and blue, respectively. The information about the highlighted function is shown in the scrolled text area just below the menu bar of the window. To highlight another step or impulse response, the user can

   1. either push the **Next Response** button,
   2. or select the function to be highlighted from the function list on the main window and push the **Display** button there.

- in the **Special** menu, the user can remove step or impulse functions from the window, or add other undisplayed step or impulse functions to the window.

All of the functionalities that a single-response window provides as described in Section 5.3.1 are also afforded by a multiple-response window.

Multiple-MSV Windows

To create a multiple-MSV window, the user (i) first selects the undisplayed MSV functions on the function list of the main window, and (ii) then pushs the **Display** button. A multiple-MSV window is shown in Figure 5.26 which looks and functions similar to Figure 5.23 except that:

- all of the responses of the functions corresponding the current design are displayed. However, only one of the responses is highlighted. The responses of the highlighted function corresponding to the last and the last 2nd designs are displayed and highlighted too. Only the bound of the highlighted function is shown with the undesired region displayed in the color of lavender. Similar to the single-function case, the responses corresponding to the current, the last and the last 2nd designs are plotted in the colors of black, red and blue, respectively. The information about the highlighted function is shown in the scrolled text area just below the menu bar of the window. To highlight another MSV response, the user can

   1. either push the **Next Response** button,
   2. or select the function to be highlighted from the function list on the main window and push the **Display** button.

- in the **Special** menu, the user can remove MSV functions from the window, or add other undisplayed MSV functions to the window.

All of the functionalities that a single-MSV window provides as described in Section 5.3.1 are also afforded by a multiple-MSV window.

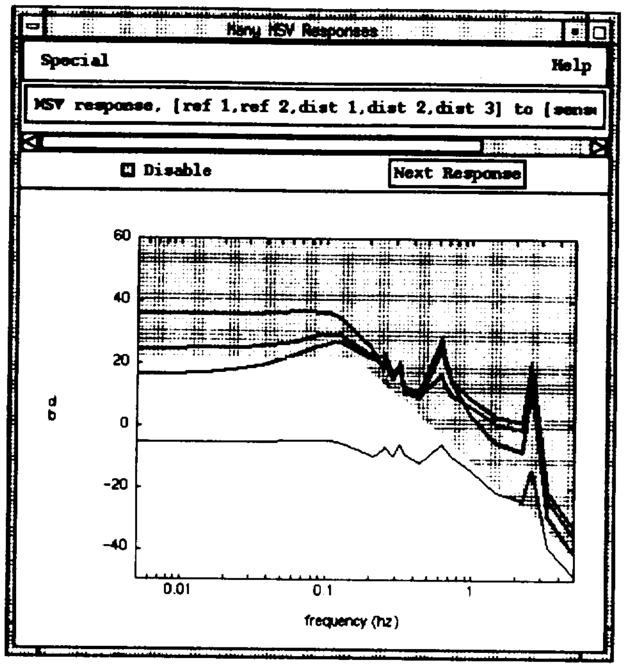


Figure 5.26: A multiple-MSV window

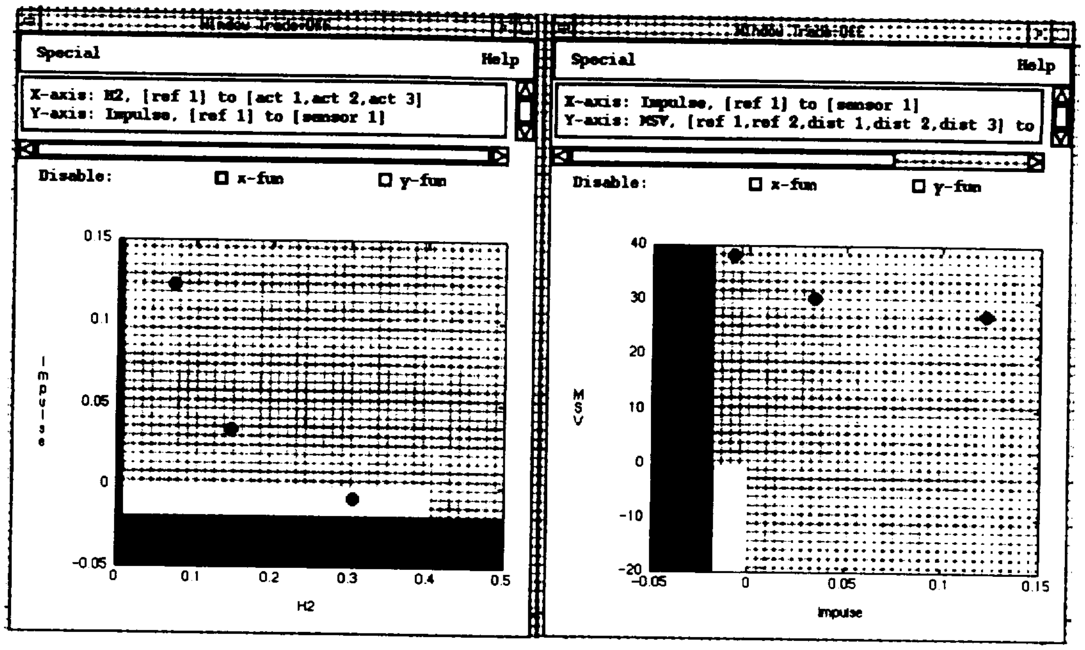
5.3.3 Trade-Off Windows

A trade-off window is created by choosing any two functions from the function list on the CODA main window, and pushing the TradeOff button. Each of the x and the y axes of the plot on the trade-off window indicates the function value associated with either of the chosen functions, which is equal to

- the RMS value for the $\mathcal{H}_2$ function (see (2.12));
- the design function value defined in (2.21) or (2.25) for the step/impulse or MSV function.

Two examples of the trade-off window are shown in Figure 5.27 where:

- there are at most 3 points shown on the trade-off window, which are marked by solid circles, and have the colors of black, red and blue respectively corresponding to the current, the last and the last 2nd designs.
- it is clear that, for a feasible design,
  1. the corresponding RMS value of an $\mathcal{H}_2$ function is bounded above by the upper bound, and bounded below by the minimum value of the function;
  2. the corresponding design function values defined in (2.21) and (2.25) for the step, impulse and MSV functions are bounded above by 0. For the step



X-axis: H2, [ref 1] to [act 1,act 2,act 3]
Y-axis: Impulse, [ref 1] to [sensor 1]

X-axis: Impulse, [ref 1] to [sensor 1]
Y-axis: MSV, [ref 1,ref 2,dist 1,dist 2,dist 3] to

Figure 5.27: Two trade-off windows

and the impulse functions, the design function values are bounded below by

$$\min\left(\frac{\bar{b}-\underline{b}}{2}\right),$$

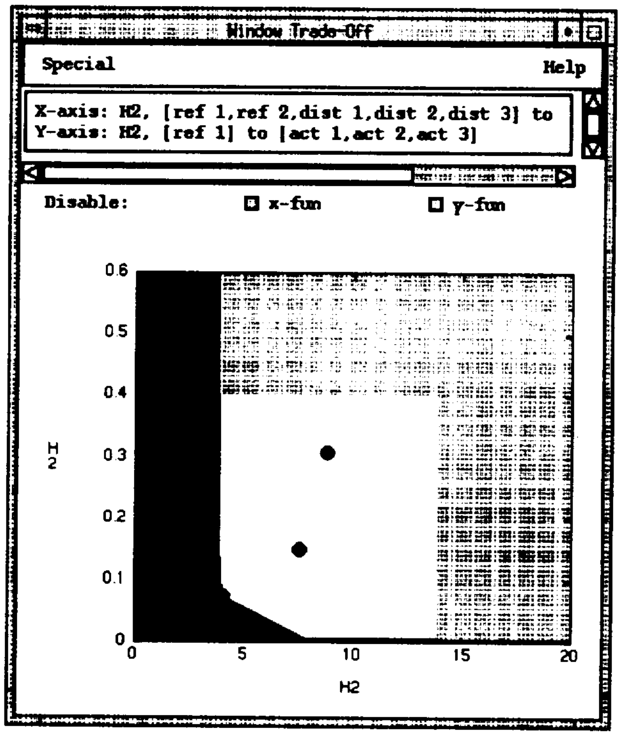
where $\bar{b}$ and $\underline{b}$ denote the upper and the lower bounds respectively (see (2.18)).

The above explains how the desired region in the white color in Figure 5.27 is formed. The gray region indicates the unreachable region, while the yellow region indicates the undesired region.

- in the Special menu, the user can
  1. exchange the two functions on the x and the y axes.
  2. close the trade-off window.
- the information of the two functions defining the trade-off plot is shown on the scrolled text area just below the menu bar.
- the status of activeness of the x-axis and the y-axis functions as parts of design specifications is controlled by two togglebuttons.
- the user can re-scale the plot by moving the cursor to any of the boundaries or corners of the plot area, and dragging it with the left button of the mouse pushed.

$\mathcal{H}_2$ Trade-Off Window

A trade-off window is called an $\mathcal{H}_2$ trade-off window if both two functions defining the trade-off window are $\mathcal{H}_2$ functions. An example of $\mathcal{H}_2$ trade-off window is shown in Figure 5.28 which looks similar to Figure 5.27 except that:



X-axis: H2, [ref 1,ref 2,dist 1,dist 2,dist 3] to
Y-axis: H2, [ref 1] to [act 1,act 2,act 3]

Figure 5.28: An $\mathcal{H}_2$ trade-off window

- a trade-off curve is shown, which is part of the boundary between the unreachable region and the reachable region. Each point on the trade-off curve corresponds to a unique design which can be acquired by pointing the cursor to the point of interest on the curve, and clicking the left button of the mouse with <Control> and <Shift> keys pushed.
- the bounds for the $\mathcal{H}_2$ functions can be changed by moving the cursor on the reachable region with the left button of the mouse and the <Control> key pushed. A widget shown in Figure 5.22 will appear for the user to confirm or cancel the change of the bounds. The undesired region is displayed in the color of lavender to indicate that the bounds of the $\mathcal{H}_2$ functions, and hence the undesired and the desired regions, can be changed on this window.

References

[1] S. Boyd, V. Balakrishnan, C. Barratt, N. Khraishi, X. Li, D. Meyer and S. Norman, "A new CAD method and associated architectures for linear controllers," *IEEE Trans. Automat. Contr.*, Vol. AC-33, No. 3, pp. 268-283, March 1988.

[2] S. Boyd, L. Ghaoui, E. Feron and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

[3] F. H. Clarke, *Optimization and Nonsmooth Analysis*. Philadelphia: SIAM, 1990.

[4] R. Fletcher, *Practical Methods of Optimizations*. Second Edition, John Wieley & Sons Ltd., 1991.

[5] J. E. Higgins, "Algorithms for Optimization-Based Computer-Aided Design," Ph. D. Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, 1989.

[6] B. von Hohenbalken, "A Finite Algorithm to Maximize Certain Pseudoconcave functions on Polytopes," *Mathematical Programming*, Vol. 9, 1975, pp. 189-206.

[7] P. Huard, "Resolution of mathematical programming with non-linear constraints by the method of centers," in: J. Abadie, ed., *Non-Linear Programming*, North-Holland, Amsterdam, 1967, pp. 207-219.

[8] F. Jarre, "An interior-point method for minimizing the maximum eigenvalue of a linear combination of matrices." *SIAM J. Control and Opt.*, Vol. 31, No. 5, pp. 1360-1377, September 1993.

[9] R. L. Kosut and M. G. Kabuli, "Graphical Interactive Control Design and Implementation Environment," ISI Final Report for SBIR Phase I project under contract number NAS1-19891, August 1993.

[10] D. G. Luenberger, *Linear and Nonlinear Programming*. Second Edition, Addison-Wesley Publishing Co., 1984.

[11] E. Polak, "On the mathematical foundations of nondifferentiable optimization in engineering design," *SIAM Review*, Vol. 29, pp. 21-89, 1987.

[12] S. Salcudean, *Algorithms for Optimal Design of Feedback Compensators*. Ph. D. Thesis, Dept. of EECS, University of California, Berkeley, California 1986.

[13] Bo Wahlberg, "System Identification Using Laguerre Models," *IEEE Trans. Automat. Contr.*, Vol. AC-36, No. 5, pp. 551-562, May 1991.

[14] Bo Wahlberg, "System Identification Using Kautz Models," *IEEE Trans. Automat. Contr.*, Vol. AC-39, No. 6, pp. 1276-1282, June 1994.

[15] D. C. Youla, H. A. Jabr and J. J. Bongiorno, "Modern Wiener-Hopf design of optimal controllers, Part II: The multivariable case," *IEEE Transactions on Automatic Control*, vol. 21, pp. 319–338, June 1976.